Tool-Aware Planning for Contact-Center Analytics: Evaluating LLMs through Lineage-Guided Query Decomposition

Varun Nathan
varun.nathan@observe.ai
Observe.AI
Bangalore, India

Shreyas Guha
shreyas.slg@gmail.com
Observe.AI
Bangalore, India

Ayush Kumar
ayush@observe.ai
Observe.AI
Bangalore, India

Abstract

We present a domain-grounded benchmark and evaluation framework for **tool-aware plan generation** in contact-center analytics, where answering a business-insights query requires decomposing it into executable steps over structured tools (Text2SQL over Snowflake), unstructured tools (RAG over transcripts), and LLM-based synthesis, with explicit depends_on relations for safe parallel execution. Our contributions are threefold: **(i)** a reference-based plan evaluation framework with two complementary views—a metric-wise evaluator spanning seven dimensions (e.g., tool–prompt alignment, query adherence) and a one-shot evaluator that compares a candidate plan against a reference plan; **(ii)** a lineage-driven data curation methodology that uses an iterative evaluator→optimizer loop to refine initial plans into high-quality plan lineages while reducing manual effort; and **(iii)** a large-scale study of 14 LLMs across model families and sizes on their ability to generate step-by-step, executable, tool-assigned plans, evaluated with and without lineage in the prompt. Empirically, LLMs continue to struggle on compound queries and on plans longer than four steps; the highest aggregate metric-wise score is **84.8** (Claude-3-7-Sonnet), while the strongest one-shot **A+** rate (Extremely Good or Very Good) is only **49.75%** (o3-mini). Lineage yields mixed overall gains but improves several strong models and often helps step executability. Overall, our results expose persistent weaknesses in tool understanding—especially tool–prompt alignment and tool–usage completeness, and, show that shorter, simpler plans remain markedly easier. The benchmark, evaluation framework, and findings provide a practical path for assessing and improving agentic planning with tools in enterprise question-answering settings[1].

[1]Public dataset (https://github.com/Observeai-Research/tool-aware-planning-dataset/) with human-annotated reference plans, plan lineages, and per-planner outputs for all 14 planners

1 Introduction and Related Works

Use case and motivation. We target contact-center Question Answering (Insights) and data analytics, where plans must orchestrate *structured* analysis via Text2SQL over Snowflake (T2S), *unstructured* evidence via RAG over transcripts, and LLM-based synthesis/reformatting under tight latency, correctness, and auditability constraints.

LLMs are increasingly used as *agents* that decompose goals into multi-step plans and invoke external tools, making *evaluation of planning quality* an important problem in its own right. Benchmarks such as Li et al., 2023 (tool selection and argument filling) and Liu et al., 2023 (agentic behavior across diverse environments) probe tool-augmented reasoning; Song et al., 2025 targets API workflows; Deng et al., 2023 studies realistic web agents; Ma et al., 2024 evaluates sequential, multi-modal tool usage; and Wu et al., 2025b explores time-constrained, multitask planning. Beyond these, Valmeekam et al., 2023 assesses abstract plan reasoning in classic domains (e.g., Blocksworld, Logistics), and Wang et al., 2024 evaluates multi-turn tool use with language feedback. Closer to our application setting, Sahu et al., 2025 introduces *InsightBench*, which evaluates business analytics agents through multi-step insight generation. Complementary lines of work examine neuro-symbolic and optimization hybrids—e.g., Obata et al., 2024 integrates dependency graphs and linear programming for multi-robot task planning.

Why contact-centers need a different planning lens. Queries in contact-centers (e.g., "How did escalation rates and QA outcomes differ by time zone?") require *tool-aware plans* that orchestrate structured data (e.g., Text2SQL over Snowflake) and unstructured data (RAG over transcripts), with *correct argument binding* (filters vs. prompts) and *explicit dependencies* for *parallel* execution under latency constraints. Errors in tool choice, place-

Proceedings of the Fifth Workshop on Generation, Evaluation and Metrics (GEM), pages 893–943

holders, dependency wiring, or prompt scope can silently degrade results, making plan *correctness*, *executability*, and *format fidelity* first-class requirements in production systems.

Limitations of existing benchmarks. Prior agent/planning benchmarks typically presume (i) *one correct tool per sub-task*, (ii) *tight I/O coupling* (step outputs directly consumable by the next), and (iii) primarily *sequential* execution. They rarely model domains where (a) *multiple tools can validly answer the same sub-question* (tool overlap), (b) outputs require *reformatting* before reuse (loose I/O coupling), and (c) concurrency is necessary for SLAs. Survey works (e.g., Cao et al., 2025; Tantakoun et al., 2025; Pallagani et al., 2024) converge on a key insight: LLMs need *structure*, *feedback*, and often *hybridization* (neuro-symbolic or optimization) to be reliable on long-horizon plans. Recent advances—Kang, 2025 (parametric problem generation; JSON + PDDL bridging), Wu et al., 2025a (cost-aware, non-sequential branching with RL), and Alidu et al., 2025 (NL→executable DAGs)—show that *structuring the planning interface* and *explicit cost/structure objectives* materially improve executability and stability. Recent work on hybrid querying and data-agent benchmarks, such as SUQL and FDABench (Liu et al., 2024; Wang et al., 2025), improves executability or final-answer quality over heterogeneous data, but does not define multi-step, tool-aware plan representations or rubrics for planning quality in contact-center settings with overlapping tools and lineage-guided revision. Complementary evaluations (e.g., Goebel and Zips, 2025) further show that LLMs struggle with *constraint compliance* and *state consistency* in complex tasks—echoing the challenges we observe when plans must satisfy strict tool-use and formatting constraints.

Gap. There is *no benchmark or evaluation protocol* that (1) targets *contact-centers* queries, (2) requires *parallel tool usage* when multiple tools (e.g., RAG and Text2SQL) contain relevant information that must be combined, (3) enforces *argument/placeholder correctness* and *dependency wiring* for *parallel* execution, and (4) captures *plan evolution* through *lineage* (intermediate, interpretable revisions) driven by an iteratively run *step-wise evaluator* and a *plan optimizer*. Existing datasets and metrics do not jointly assess *tool–prompt alignment*, *step executability*, *format/placeholder correctness*, *dependency correctness*, *redundancy*, and *tool-usage completeness*, nor do they relate these to a one-shot *plan-to-reference* comparison that measures structural closeness (precision/recall/F1).

Our approach and contributions. We study tool-aware plan generation for *contact-centers* with a fixed triad of tools: **T2S** (Text-to-SQL over Snowflake), **RAG** (transcripts), and **LLM** (synthesis/reformatting). Our contributions are:

1. **Dual-perspective evaluation framework:** Two complementary approaches of scoring a plan: (i) *7-fold metric-based* scoring and aggregation (0–100) and (ii) a *one-shot* plan-to-reference comparison (precision/recall/F1 + format) with a 7-point quality rating.

2. **Curation for lineage-driven planning:** An iterative loop with a *step-wise evaluator* and a *plan optimizer* that iteratively revises plans, improving quality over one-shot generation and *reducing human annotation*. While the dataset is proprietary, we release the *schema, prompts, and methodology*.

3. **Model study:** Results for *14 LLMs* spanning architectures, sizes, and context lengths (e.g., o3-mini (OpenAI, 2025b), GPT-4o/mini (OpenAI et al., 2024), Claude-3-7-Sonnet/3-5-Haiku (Anthropic, 2024), Llama variants (Research, 2024; Meta, 2025), Nova family (Intelligence, 2024)), with analyses by *subjectivity*, *compoundness*, and *plan length*, plus the effect of *plan-lineage prompting*.

2 Task Formalization

Problem setup. Given a fixed tool set T and a natural-language query Q, a planner must return an *executable, tool-aware plan* P.

Plan schema. We represent a plan as an ordered sequence of steps:

$$P = \langle s_k \rangle_{k=1}^{n}, \quad s_k = (t_k, p_k, D_k)$$

where n is the number of steps, $t_k \in T$ is the tool chosen for step k, p_k is the tool instruction (prompt) for step k, and $D_k \subseteq \{1, \ldots, k-1\}$ is the set of dependency indices indicating which prior steps s_i must finish before s_k can execute. Since dependencies may only point to earlier steps, the induced graph is required to be acyclic (a DAG), enabling safe parallel execution of independent steps. Before scoring, we run a simple validator that checks index validity and rejects non-acyclic plans. In practice, plans are materialized as JSON objects whose keys are step indices and whose values store `query` and `depends_on` fields.

Plan lineage. For a query Q_i, our curated data stores a *plan lineage*, i.e., an ordered list

$$\mathcal{L}(Q_i) = \langle P_i^{(0)}, P_i^{(1)}, \ldots, P_i^{(M_i)} \rangle,$$

where $P_i^{(0)}$ is the initial (typically weakest) plan and $P_i^{(M_i)}$ is the best (reference) plan. Each subsequent plan is produced from the previous by an evaluator→optimizer loop that corrects tools, prompts, and dependencies (see §3).

Tool set. We use three internally built tools (full details in Appendix A):

- **T2S (Text-to-SQL over Snowflake):** answers queries using structured contact-center data (e.g., call drivers, key moments during interactions, Qualtiy-Assurance metrics).
- **RAG (Retrieval-Augmented Generation):** answers queries using customer–agent transcripts.
- **LLM (Synthesis/Reformatting):** composes, reformats, and aggregates outputs (e.g., extracting `call_ids` from RAG tables; merging multi-tool evidence).

Execution constraints. Given dependencies D_k, step s_k may only reference prior outputs via placeholders (e.g., "(3)") and must not redundantly refilter on criteria already encoded by its dependency inputs.

Listing 1: Example query-plan pair

```
# Query: Compare QA scores for professionalism
    and resolution procedures in unresolved
    calls where sentiment transitioned from
    negative to positive.
# Plan:
{
  "1": {"query": "T2S([], 'Fetch
      interaction_ids of unresolved calls')",
      "depends_on": []},
  "2": {"query": "RAG((1), 'Fetch calls where
      the sentiment transitioned from negative
      to positive within the transcript')",
      "depends_on": [1]},
  "3": {"query": "LLM('Extract interaction_ids
      from Data Insights in (2).')",
      "depends_on": [2]},
  "4": {"query": "T2S((3), 'Retrieve QA scores
      for resolution procedures in these
      calls.')", "depends_on": [3]},
  "5": {"query": "T2S((3), 'Retrieve QA scores
      for professionalism in these calls.')",
      "depends_on": [3]},
  "6": {"query": "LLM('Compare QA scores from
      (4) vs. (5) in light of unresolved status
      and sentiment transitions.')",
      "depends_on": [4,5]}
}
```

Example lineage (excerpt). Due to space limits we show and describe the final plan; full lineage appears in Appendix B.

Final Plan (best): the 6-step plan (shown above) separates filtering for unresolved calls (T2S) from within-call sentiment shift (RAG), extracts IDs (LLM), and aggregates QA metrics (T2S) before synthesis.

3 Dataset Generation Methodology

A dataset of queries and plan lineages is essential for evaluating the performance of both proprietary and open-source, instruction fine-tuned, out-of-the-box, LLMs on the task of plan generation. Our dataset is built via a four-stage pipeline: ① controlled *query generation*; ② one-shot *initial plan generation* with prompt engineering; ③ an *iterative evaluator→optimizer* loop that produces a *plan lineage* per query; and ④ *human verification* of the final plans.

① Query generation. We use **GPT-4o** to synthesize queries along two axes: (i) *subjectivity* (objective vs. subjective), and (ii) *compoundness* (simple vs. compound). This ensures coverage across measurable vs. interpretive asks, and single vs. multi-ask structures. See section C.1 for the description of each dimension, Tables 27 & 38 for the prompts used and examples for different query types.

② Initial plan generation. Plans are produced one-shot by an LLM using a two-part prompt (system: task + tool schema; user: formatting + examples). We compare low/medium/high tool-detail variants and vary few-shot counts; the *medium-detail* prompt with 6–8 examples yields the best accuracy without overload. See section C.2 for more details and Table 27 for the prompts used.

③ Iterative Evaluator→Optimizer Feedback Loop. We refine one-shot initial plans via a lightweight, non-executing feedback loop (Fig. 1). Each *pass* freezes the current plan P and iterates once over all its steps. At each step, the *Step-wise Evaluator* inspects the step's tool, prompt, and declared dependencies, emitting one or more diagnostic tags (e.g., INCORRECTTOOL, COMPLEXPROMPT, REPEATEDDETAIL, MULTITOOLPROMPT, INCORRECTPROMPT, NOCHANGE). The *Plan Optimizer* consumes these tags but may choose *not* to revise the step—even when the evaluator suggests a change—if a local edit would harm global coherence. When it does edit, it applies

local repairs (*Change 0*) followed by global coherence fixes (*Change 1*) to maintain a valid DAG (placeholders and dependency closure), optionally splitting/merging steps.

Let P' denote the optimizer's output after processing the current step. We append to the *plan lineage only if* $P' \neq P$, then set $P \leftarrow P'$; otherwise we proceed to the next step with P unchanged. A pass completes after every step is visited exactly once. The loop stops if (i) an entire pass makes no change (the plan at the end of the pass equals the plan at the start), or (ii) the number of passes exceeds `max_passes` $= 4$. Because we record only distinct plans, the lineage is an ordered sequence from the weakest to the best plan. The loop never executes tools; it improves plan *text* only, enabling scalable dataset curation without runtime calls. Pseudocode appears in Appx. C.3, Alg. 1, with termination in Lemma 1 and prompts used in Tables 28 – 29.

④ **Human verification.** Final lineage heads are reviewed by expert annotators. Minor fixes (if any) are applied to certify the *best* plan. The lineage (all intermediate plans) is retained for analysis and training.

Evaluation Metadata. After obtaining the human-verified best plan (Step 4), we derive lightweight *metadata* from both the lineage and the final plan to study LLM performance on plan generation. These features are computed without executing tools and are stored with each example to enable controlled analyses and stratified reporting. Concretely, we record: (i) lineage length (distinct plans), (ii) number of passes until convergence, (iii) per-pass revision count, (iv) step count, and (v) *Number of Hops*. See Table 51 for recorded stats.

Number of Hops We compute a query's *number of hops* from its best plan's dependency graph (Refer C.6 for details on definition and computation). A hop is the level of dependency needed to produce the final answer: *zero-hop* plans have no dependencies (direct tool calls yield the answer); *one-hop* plans have exactly one dependency layer (e.g., an LLM step depending on two independent producers); *two-hop* plans have two sequential dependency layers (e.g., a final LLM step depends on RAG/T2S steps that themselves depend on a T2S filter); *three-plus* plans have three or more sequential dependency layers. We use this hop count as metadata for stratifying LLM performance.

Extended details. See Tables 35–37 for an end-to-end lineage from **Query** → **Step-Wise Evaluator / Plan Optimizer** outputs → **Final Plan**.

4 Plan Evaluation Methodology

We evaluate plans in two complementary modes. First, a *reference-based metric-wise* framework yields a 0–100 score by aggregating seven targeted metrics grouped into *Effectiveness* (70%) and *Efficiency* (30%). Second, a *reference-based one-shot evaluator* measures closeness to a best-possible plan using step-level Precision/Recall/F_1 and maps the result to a seven-point rating.

Overall Score (Metric-Wise Framework). Let $\mathcal{E}$ be effectiveness metrics and $\mathcal{F}$ be efficiency metrics, with per-metric scores $m_k \in [0, 1]$ and weights w_k s.t. $\sum_{k\in\mathcal{E}} w_k = 0.7$ and $\sum_{k\in\mathcal{F}} w_k = 0.3$. The overall score is:

$$\textsc{Score}(P) \;=\; 100 \cdot \Big(\sum_{k\in\mathcal{E}} w_k m_k \;+\; \sum_{k\in\mathcal{F}} w_k m_k \Big).$$

4.1 Metric-Wise Evaluation

Effectiveness (70 pts). Assesses correctness and executability.
(i) Tool–Prompt Alignment (20 pts): tool capabilities match prompt intent.
(ii) Format Correctness (20 pts): JSON parseable; valid placeholders, quotes, parentheses.
(iii) Step Executability / Atomicity (15 pts): each step is a single, atomic operation.
(iv) Query Adherence (15 pts): if executed perfectly, the plan fully answers the query.

Efficiency (30 pts). Assesses optimality.
(v) Dependencies (10 pts): correct, minimal `depends_on`; consistent placeholders.
(vi) Redundancy (10 pts): no duplicated work; no repeated filters causing unnecessary joins.
(vii) Tool-Usage Completeness (10 pts): when a task clearly benefits from two tools in sequence or parallel (e.g., T2S *and* RAG), the plan includes both.

Evaluator pipeline. All seven metrics are reference-based and scored by specialist LLM evaluators with rubric prompts (details in Tables 30–32). The detailed metric definitions and scoring methodologies are provided in Section D.1.
Metric Weighting. We learn per-metric weights on the validation set to enforce monotonic improvement across the top three plans in each lineage (best

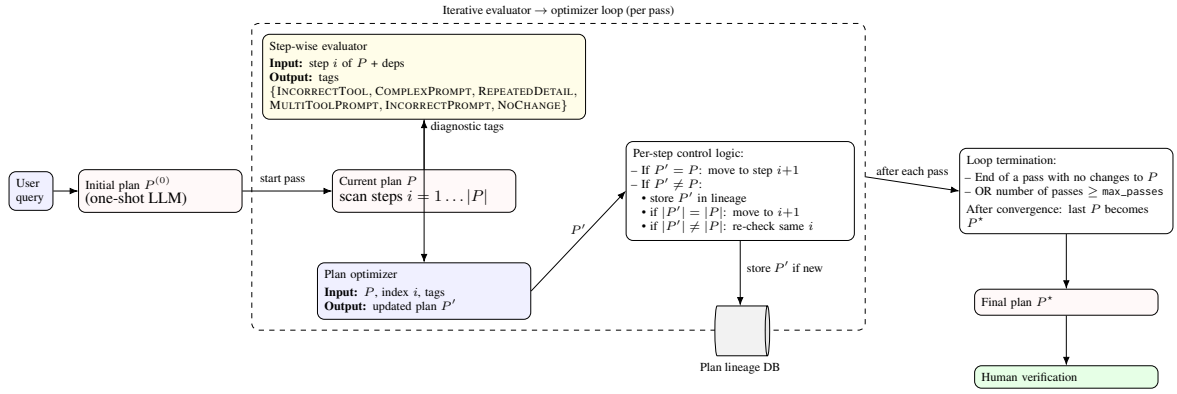

Figure 1: Iterative step-wise evaluator → plan optimizer loop producing a plan lineage. For each pass, every step i of the current plan P is diagnosed by the step-wise evaluator and optionally edited by the plan optimizer. Any updated plan P' is appended to the lineage database. The loop stops when a full pass yields no changes or when the maximum number of passes is reached, and the final plan $P^\star$ is human-verified.

> penultimate > antepenultimate), under fixed group budgets (Effectiveness = 0.7, Efficiency = 0.3); see Appendix §D.4 for details.

4.2 One-Shot Overall Evaluation

Given candidate plan P and best plan $P^\star$, the Judge LLM is prompted to compute Precision, Recall, and F_1 from step matches and to assess Format Correctness, Dependencies, and Placeholders solely on P. We map F_1 to a seven-point rating: *Extremely Good* (>95%), *Very Good* (>85%), *Good* (>75%), *Acceptable* (>60%), *Bad* (>45%), *Very Bad* (>30%), *Extremely Bad* ($\leq$ 30%). If P is not JSON-parseable, we assign *Extremely Bad* regardless of F_1. Full prompt and output schema are provided in Table 33 and Section D.2 respectively.

5 Experimental Setup

Data splits and usage. We curate 600 contact-center queries and their plan lineages using the iterative Evaluator→Optimizer loop (Sec. 3 and Appx. C). We stratify by subjectivity, and compoundness. Splits: *Train* (20) for prompt construction, *Validation* (80) for module tuning (metric-wise evaluators, one-shot Judge, step-wise evaluator, plan optimizer), and *Test* (500) for model benchmarking. Due to proprietary constraints, the dataset cannot be released.[2] A detailed, step-by-step description of how we sampled, annotated, validated, and finalized lineages and gold plans is provided in E.1.

Plan generation models. We benchmark 14 LLMs for one-shot plan generation under two prompts: *without lineage* and *with lineage* (the latter includes per-query plan lineage examples sourced via our feedback loop). Models: *Claude-3-7-Sonnet/Claude-3-5-Haiku (Anthropic, 2024), Claude-Sonnet-4 (Anthropic, 2025), Nova-Pro/Nova-Lite/Nova-Micro (Intelligence, 2024), Llama3-2-1B-Instruct/Llama3-2-3B-Instruct/Llama3-70B-Instruct (Research, 2024), Llama4-Maverick-17B-Instruct (Meta, 2025), GPT-4o/GPT-4o-Mini (OpenAI et al., 2024), GPT-4.1-Nano (OpenAI, 2025), o3-Mini (medium reasoning) (OpenAI, 2025b)*. Model configurations and prompt templates are in E.5 and Tables 50 & 27.

Evaluation models. Both evaluators are LLM-based. We use *Claude-Sonnet-4* as the Judge LLM for (i) the one-shot overall evaluation (Precision/Recall/F_1 + 7-point rating) and (ii) all seven metric-wise evaluators (Sec. 4). The same rubric prompts are used across splits (D.1, D.2).

Feedback-loop modules. For the iterative plan lineage construction, we use *GPT-4o* for both the Step-wise Evaluator and the Plan Optimizer (C.4). We cap passes via a single hyperparameter `max_passes` for which we adopt **4** (empirically balances gains vs. latency) in our runs.

Implementation. All generation, evaluation, and feedback loops are implemented in Python on local infrastructure. Model access uses Bedrock and

[2]We provide full prompts, and scoring rubrics to facilitate replication on non-proprietary corpora; see Tables 27–33 and E.7. We also release a separate public 200-query dataset (`https://github.com/Observeai-Research/tool-aware-planning-dataset/`) built from LLM-generated queries modeled on a synthetic test account, with human-annotated reference plans, plan lineages, and per-planner outputs for all 14 planners under both prompt settings; see Appx. E.2 for details.

LiteLLM; we fix random seeds and use deterministic decoding for evaluators. Infra, seeds, rate limiting, and caching are detailed in E.7.

Human agreement. All human annotations used for tuning and validating evaluators exhibit substantial agreement; detailed protocol and κ statistics (with CIs) are reported in E.3.

6 Results

6.1 Overall Plan Generation Quality

One-shot evaluator (reference-based). Table 1 shows the proportion of plans in each quality tier: *A+* (Extremely or Very Good), *A* (*A+* + Good), and *B* (*A* + Acceptable) for each model and prompt setting, with and without lineage. Overall performance is modest: the best result is **o3-mini** with **49.75%** in *A+* *without* lineage, indicating that even strong models struggle to reliably produce near-gold plans. **o3-mini** also leads the *A* bucket at **65.99%** (no lineage), while **GPT-4o** attains the highest *B* coverage at **82.74%** (no lineage). Other models—including *Claude-Sonnet-4*, *Claude-3-7-Sonnet*, and *Claude-3-5-Haiku* mostly remain in the low-to-mid 30% range for *A+*, and smaller *Llama3* variants (1B/3B) are single-digit ($\sim 5.6\%$).

Effect of lineage in prompt. Of 14 LLMs, 6 improve, 5 degrade, and 3 show no change on the *A+* bucket when lineage is included, yielding no clear overall benefit.

Metric-wise evaluator (reference-informed). Tables 2–3 present aggregate scores (in the range of 0-100) and per-metric results. **Claude-3-7-Sonnet** is highest overall at **84.8** (with lineage), followed by **llama4-maverick-17b-instruct** at **82.5** and **GPT-4o** at **82.82** (both without lineage). Across models, average *Format* score (with lineage) is $\mathbf{14.09/20}$—six models fall below this, indicating non-trivial formatting challenges. Average *Tool-Prompt Alignment* is $\mathbf{12.87/20}$ (with) and $\mathbf{13.82/20}$ (without), suggesting many models still misapply tools/prompts. Average *Tool Usage Completeness* is $\mathbf{4.87/10}$ (with) and $\mathbf{7.11/10}$ (without), reflecting difficulty recognizing when both T2S and RAG are jointly warranted. On average, prompts *without* lineage score higher overall; only 3/14 models improve overall with lineage. An exception: for *Step Executability*, 9/14 models improve with lineage despite lower mean due to outliers (e.g., larger gains for *Llama3-2-1B* in the no-lineage setting).

Sensitivity to metric weights. Our aggregate "learned" score is derived from metric-wise evaluator outputs and weights learned from human-preferred lineage triples (App. D.4). To test whether planner rankings are overly sensitive to these weights, we recompute scores under (i) an equal-weights scheme over normalized metrics and (ii) ten random weight vectors that preserve the 70:30 Effectiveness–Efficiency budget via Dirichlet draws. For both prompt settings (with-lineage and without-lineage), planner rankings remain highly stable: Spearman rank correlation (Zar, 2005) between the learned and equal-weights rankings is $\rho = 0.934$ (with-lineage) and $\rho = 0.894$ (no-lineage), while random-weight rankings achieve median correlations of $\rho = 0.890$ and $\rho = 0.842$ respectively (Table 39; App. D.5). This empirical analysis shows that planner-level conclusions do not depend on a single choice of metric weights.

Key grouped takeaways (details in Appendix F). (i) **Simple vs. Compound**: clear advantage for *Simple* across both evaluators and both prompt settings. (ii) **Plan Length**: queries whose best plans have $[1, 4]$ steps consistently outperform those requiring $[5, 15]$ steps. (iii) **Objective vs. Subjective**: mixed for one-shot (slight edge for Objective without lineage); metric-wise shows a counterintuitive tilt toward Subjective for many models. (iv) **#Hops**: no stable pattern across models.

Correlation with end-to-end QA quality. Although our benchmark is non-executing by design, we conducted a small end-to-end study on 200 LLM-generated test queries, comparing the no-plan baseline (**R1**), a north-star system that executes *human-annotated reference plans* (**R2**), and the same stack driven by *LLM-generated plans* (**R3**). Final answers are scored by an in-house Judge LLM using a four-metric rubric (Validity, Consistency, Completeness, Redundancy; App. D.6). R2 achieves a win rate of **58.7%**, versus **42.75%** for R1 and **33.33%** for R3, and reference plans also obtain substantially higher planning scores than LLM-generated plans, supporting a positive correlation between planner quality and end-to-end QA quality.

6.2 Effectiveness of the Iterative Evaluator→Optimizer Loop

We quantify the impact of our Iterative Evaluator→Optimizer loop using the one-shot (reference-based) evaluator, comparing the

LLM	With Lineage			Without Lineage		
	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)
o3-mini	43.15	53.30	72.59	**49.75**	65.99	80.20
gpt-4o	**45.69**	62.44	81.22	41.12	57.87	82.74
gpt-4o-mini	31.98	48.73	64.47	31.98	52.28	71.57
claude-3-5-haiku	24.87	45.69	62.44	**30.46**	47.72	72.59
claude-sonnet-4	**30.46**	52.79	73.60	29.95	49.75	74.11
llama4-maverick-17b-instruct	20.30	36.55	54.31	**20.81**	40.61	62.44
nova-pro	18.27	42.64	61.93	**19.80**	41.62	59.39
claude-3-7-sonnet	**30.46**	48.22	70.56	19.80	39.09	69.04
llama3-70b-instruct	**18.78**	44.16	59.90	17.26	35.53	55.33
gpt-4.1-nano	**17.26**	28.43	55.33	16.24	30.46	53.30
nova-micro	**20.81**	35.53	53.30	15.74	37.06	53.81
nova-lite	13.71	30.96	47.24	13.71	34.52	47.72
llama3-2-3b-instruct	5.08	10.66	17.26	**5.58**	11.68	26.90
llama3-2-1b-instruct	0.00	0.00	0.00	0.00	0.00	1.52
Grand Total (#)	**500**			**500**		

Table 1: Plan generation quality comparison using prompts **with and without lineage**, evaluated with the **one-shot evaluator** on test data. The highest score in the **Extremely Good, Very Good** bucket is highlighted in green; blue indicates better performance with lineage, and magenta indicates better performance without lineage.

LLM	**Overall** [0-100]	**Format** [0-20]	**Tool Prompt Align.** [0-20]	**Step Exec.** [0-15]	**Query Adhr.** [0-15]	**Depend.** [0-10]	**Redund.** [0-10]	**Tool Usage Compl.** [0-10]
claude-3-7-sonnet	**84.8**	**18.46**	**15.32**	12.61	12.6	9.78	8.97	7.07
llama4-maverick-17b-instruct	82.26	17.14	14	13.09	12.69	9.35	9.41	6.59
gpt-4o	81.47	16.42	13.77	13	12.85	9.66	8.58	7.2
claude-sonnet-4	79.9	12.89	14.68	13.18	12.35	9.68	8.59	**8.54**
llama3-3-70b-instruct	79.77	17.04	14.52	13.54	12.1	9.55	9.36	3.66
nova-pro	79.24	15.51	14.57	**14.32**	12.31	**9.81**	9.36	3.35
nova-micro	77.28	18.25	13.42	12.92	12.08	9.58	**9.44**	1.59
gpt-4o-mini	77.26	14.06	14.54	13.58	12.34	8.99	9.12	4.63
gpt-4.1-nano	77.16	13.48	13.88	13.36	11.92	9.09	8.79	6.65
o3-mini	71.85	10.9	14.79	10.91	**13.58**	8.68	9.32	3.66
claude-3-5-haiku	71.27	12.62	12.77	11.6	11.37	8.42	8.14	6.34
nova-lite	70.53	13.84	12.92	12.58	11.93	8.37	9.42	1.46
llama3-2-3b-instruct	70.51	16.62	10.94	10.65	9.74	9.23	9.07	4.27
llama3-2-1b-instruct	20.27	0	0.07	0	10.5	4.23	2.24	3.23
Average (Normalized)	**73.11**	**70.43**	**64.36**	**78.73**	**80.18**	**88.88**	**85.56**	**48.74**

Table 2: Plan generation quality using prompts **with lineage**, evaluated with the **metric-wise evaluator** on test data (**500** queries). The **Normalized Average** in the last row shows the average per metric normalized by that metric's maximum score. Highest scores per metric are highlighted in blue, and the three metrics with the lowest normalized scores are highlighted in red.

initial Nova-Lite plans (*pre-loop*) to the *final* plans produced by the loop (*post-loop*). Table 4 buckets plans into *Extremely Good*, *Very Good*, *Good*, *Acceptable*, *Bad*, *Very Bad*, and *Extremely Bad*.

Overall. Post-loop plans improve the top buckets: **Extremely Good** rises from **4.4%** (pre) to **8.0%** (post), and **Very Good** from **9.2%** to **14.8%**. Although *Good* drops from **20.8%** to **16.8%**, the net shift toward higher-quality brackets demonstrates the loop's effectiveness at elevating plan quality.

By query traits (details in Appendix G). Gains are consistent across (i) Objective/Subjective, (ii) Simple/Compound, and (iii) hop counts (0/1/2/3+). Improvements are largest for **Objective** queries and **Simple** queries, and - by relative margin - for **3+ hop** plans, indicating the loop particularly helps longer-horizon compositions.

LLM	Overall [0-100]	Format [0-20]	Tool Prompt Align. [0-20]	Step Exec. [0-15]	Query Adhr. [0-15]	Depend. [0-10]	Redund. [0-10]	Tool Usage Compl. [0-10]
claude-3-7-sonnet	**83.33**	17.36	15.48	11.46	**13.37**	9.63	9.08	6.95
llama4-maverick-17b-instruct	82.5	17.35	13.59	12.67	12.35	9.77	8.84	7.93
gpt-4o	82.82	15.97	14.6	12.84	12.78	9.75	8.59	8.29
claude-sonnet-4	77.98	10.91	**16.67**	11.78	12.91	9.7	8.7	7.32
llama3-3-70b-instruct	81.11	15.25	14.59	13.32	12.34	9.59	9.08	6.95
nova-pro	82.7	15.92	14.27	**14.11**	12.69	9.78	8.86	7.07
nova-micro	82.07	**17.9**	13.25	13.44	12.13	9.5	8.65	7.2
gpt-4o-mini	82.78	16.78	13.72	13.97	12.27	9.62	8.54	7.88
gpt-4.1-nano	76.05	12.92	13.53	12.76	11.98	8.93	8.13	7.8
o3-mini	74.06	12.28	14.89	10.56	13.09	9.06	**9.42**	4.76
claude-3-5-haiku	82.06	14.67	15.92	14.01	12.11	**9.79**	9.14	6.43
nova-lite	75.14	15.27	13.09	13.05	11.9	8.88	9.3	3.66
llama3-2-3b-instruct	75.84	16.33	10.8	11.98	11.64	8.81	7.74	8.54
llama3-2-1b-instruct	56.79	9.03	9.07	8.86	7.56	7.09	6.34	**8.84**
Average (Normalized)	78.23	**74.26**	**69.10**	**83.24**	**80.53**	**92.78**	**86.01**	**71.12**

Table 3: Plan generation quality using prompts **without lineage**, evaluated with the **metric-wise evaluator** on test data (**500** queries). The **Normalized Average** in the last row shows the average per metric normalized by that metric's maximum score. Highest scores per metric are highlighted in blue, and the three metrics with the lowest normalized scores are highlighted in red.

Tag	Pre-Loop		Post-Loop	
	#	%	#	%
Extremely Bad	28	5.60	23	4.60
Very Bad	127	25.40	109	21.80
Bad	107	21.40	122	24.40
Acceptable	66	13.20	48	9.60
Good	104	**20.80**	84	16.80
Very Good	46	9.20	74	**14.80**
Extremely Good	22	4.40	40	**8.00**
Grand Total	**500**	**100.00**	**500**	**100.00**

Table 4: Effectiveness of the **iterative Evaluator→Optimizer loop**, measured using the one-shot evaluator on test data. Higher values between **Pre-Loop** and **Post-Loop** in the **Good, Very Good, Extremely Good** buckets are highlighted in green.

6.3 Module Validation on Validation Set

We validate the four core evaluators/modules on the held-out validation split (Sec. E.1).

Metric-wise evaluator. We compare a *Single* (all 7 metrics in one prompt) vs. *Deconstructed* (one prompt per metric) setup, each in *reference-free* and *reference-based* modes. For each query, we form triplets of the three highest-quality plans in its lineage and assess *relaxed triplet ranking agreement* with human inequalities per metric (Table 21). The deconstructed, reference-based setting performs best across all seven metrics, with $> 90\%$ agreement for DEPENDENCY, FORMAT, TOOL USAGE COMPLETENESS, and $> 80\%$ for QUERY ADHERENCE, REDUNDANCY, TOOL-PROMPT ALIGNMENT; STEP EXECUTABILITY attains 79.43%. Refer to section H.1 for more details on the setup.

One-shot overall evaluator. We report macro Precision/Recall/F1 for label agreement over seven quality tags (*Extremely Bad* → *Extremely Good*) between the LLM judge and humans (Table 22). The macro averages are $0.92/0.93/0.92$, with every tag at F1 ≥ 0.85. Refer to section H.2 for more details on the setup.

Step-wise evaluator. On $N{=}400$ step instances (80 queries $\times$ $\approx$5 steps/plan), multi-label tag agreement (Precision/Recall/F1) appears in Table 25. Macro F1 is 0.84; INCORRECT PROMPT and REPEATED DETAIL reach 0.91 F1, while NO CHANGE is hardest at 0.75. Refer to section H.3 for more details on the setup.

Plan optimizer. Using the tuned one-shot judge to compare optimizer revisions against human gold for 160 pairs (Table 26), 74.5% of optimizer out-

puts land in the top buckets (*Extremely Good* 28.13%, *Very Good* 24.38%, *Good* 21.88%), and 10.63% are *Acceptable*. Refer to section H.4 for more details on the setup.

Judge robustness across models. Additionally, to mitigate the risk of single-judge bias, we replicated the validation and a small-scale test analysis with an alternative judge (GPT-5 (OpenAI, 2025a)) in the reference-based, deconstructed configuration. On the validation set, both Sonnet-4 and GPT-5 achieve high triplet-ranking agreement with humans for the metric-wise evaluator (e.g., $>$ 90% for DEPENDENCY, FORMAT, REDUNDANCY, TOOL USAGE COMPLETENESS; Tab. 21), and Sonnet-4 attains slightly higher macro F1 as a one-shot judge than GPT-5 (0.921 vs. 0.882; Tabs. 22–23). On a test subset of 50 queries per planner (700 query–planner pairs across 14 planners), planner rankings under Sonnet-4 and GPT-5 are strongly correlated for the overall metric-wise score (ρ=0.60) and for most individual metrics (e.g., DEPENDENCY ρ=0.84, QUERY ADHERENCE ρ=0.79; Tab. 24), confirming that our conclusions are robust to the choice of judge model.

Takeaway. The deconstructed, reference-based metric suite is reliable; the one-shot judge shows strong label fidelity; the step-wise evaluator identifies actionable errors; the plan optimizer substantially improves plan quality relative to initial drafts. Additional experiments with GPT-5 as an alternative judge (Appx. E.4) show similar human alignment and strongly correlated planner rankings, indicating that our conclusions are robust to the choice of judge model.

7 Conclusion

We present a domain-grounded framework for *tool-aware plan generation* in contact-centers that unifies (i) a formal, executable plan schema with explicit `depends_on` for parallelism, (ii) a tool interface spanning structured (T2S/Snowflake) and unstructured (RAG/transcripts) evidence plus LLM synthesis, (iii) an iterative evaluator→optimizer loop that yields **plan lineage**, and (iv) a two-track evaluation methodology (metric-wise and one-shot, reference-based). Beyond a benchmark, this constitutes a practical recipe for designing, critiquing, and improving planning agents in this domain.

Empirically, a 14-LLM study shows that - even with careful prompting - models struggle with query adherence and tool-usage completeness; simpler queries and shorter plans remain markedly easier. Practically, our framework supports higher-fidelity, auditable analytics by exposing where tool choices, prompts, or dependencies fail, and by encoding safe parallel execution. Importantly, **plan lineage** is not only interpretable evidence for debugging but also a training signal: it can supervise or reward better planners via SFT or RLVR. Finally, our current offline evaluator→optimizer loop naturally sets the stage for **online, tool-aware replanning**: adding a Step Executor to form an executor→evaluator→optimizer triad enables real-time plan updates conditioned on actual tool outputs.

8 Limitations

Domain scope. The study is focused on contact-centers with domain-specific tools and schemas; we do not measure transfer to other enterprise verticals.

Proprietary vs. public data. The main 600-query benchmark used for the core results is based on production-style queries and remains proprietary due to contractual restrictions. We release a separate, anonymized 200-query public dataset built from LLM-generated queries on a synthetic test account.

Tool palette and interfaces. We fix the palette (T2S, RAG, LLM) and I/O conventions. This simplifies evaluation, but does not test broader tool ecosystems, alternative schemas, or settings where tool capabilities and interfaces evolve over time. Exploring such variants is left for future work.

LLM-based judging. Both evaluators are LLM-driven; while validated against human annotations, residual judge bias, prompt sensitivity, and dependence on a single primary judge may remain.

Offline scope and cost/latency control. Although plans encode parallelism, our benchmark is intentionally non-executing: it evaluates plan quality without running tools, and therefore does not capture runtime failures, execution-time recovery, or explicit cost- and latency-aware scheduling.

Future directions. (i) **Public proxy tasks:** extend the released synthetic dataset into broader, domain-agnostic plan-generation suites so that others can benchmark planners and judges in the open. (ii) **Learning from lineage:** use lineage traces for

SFT and RLVR to teach planners revision policies and robust tool selection. (iii) **Cost-aware planning:** integrate resource/budget objectives and dynamic branching for efficient execution. (iv) **Neuro-symbolic hybrids:** combine LLM planning with classical verification/optimization to enforce constraints. (vi) **Online replanning:** instantiate the executor→evaluator→optimizer loop to update plans at runtime based on tool results, with safeguards for rollback and partial recomputation.

9 Ethics Statement

This work studies the evaluation of tool-aware planning for contact-center analytics in an enterprise setting. Although the paper focuses on plan generation and evaluation rather than direct deployment to end users, the intended application domain involves customer-service interactions and business decision support, which raises important ethical considerations.

Privacy and confidentiality. The target domain of this work involves customer–agent conversations, enterprise analytics records, and user queries, all of which may contain sensitive or proprietary information. In particular, real enterprise queries may reveal confidential business intent, internal KPIs, operational incidents, account-specific terminology, or other non-public context even when they do not contain explicit personally identifying information. For this reason, the primary 600-query benchmark used for all reported results in the main paper cannot be publicly released. Both the internal 600-query benchmark and the public 200-query release use GPT-4o to synthesize queries; however, the public release is generated against a sandboxed synthetic test account so that no customer-specific business logic or identifiers appear, while preserving the same schema, tool palette, and evaluation setup. More broadly, any deployment of similar systems should follow strict data-governance practices, including minimization of retained data, access controls, auditing, and compliance with applicable privacy requirements.

Risk of erroneous planning. Our system evaluates plans that orchestrate tools over structured and unstructured enterprise data. Incorrect plans may lead to incomplete, misleading, or improperly grounded business insights. In operational settings, such errors could affect downstream analyses, managerial decisions, or product behavior. A central motivation of this work is therefore to make planning errors more visible and diagnosable through explicit metrics such as query adherence, tool–prompt alignment, dependency correctness, and tool-usage completeness. We view such evaluation as a safeguard, not a guarantee of correctness.

Automation and human oversight. The benchmark should not be interpreted as advocating fully autonomous decision-making in high-stakes customer-support settings. Rather, the work aims to improve the reliability and inspectability of planning components used within broader analytic workflows. Human oversight remains important, especially when generated plans or derived insights may influence quality assurance, operational reporting, or customer-impacting decisions.

Bias and representational limitations. Because the study is conducted in a proprietary contact-center environment with domain-specific tools and schemas, the resulting benchmark may reflect domain-specific assumptions, organizational practices, and data distributions. This can limit generalization and may encode biases present in the source data or existing business processes. We do not claim that the benchmark is representative of all contact centers or enterprise analytics settings.

LLM-based evaluation. Our evaluation framework relies substantially on LLM-based judges and evaluators. While these were validated against human annotations, such evaluators can still inherit biases, exhibit prompt sensitivity, or make systematic errors. We therefore treat them as structured approximations to human judgment rather than as infallible arbiters. Their outputs should be interpreted in conjunction with validation evidence and task-specific limitations.

Use and misuse. The framework is intended to support research on more reliable and transparent planning for enterprise question answering. It should not be used to justify unmonitored automation or consequential decision-making without appropriate review, governance, and validation. In particular, outputs from systems built on such methods should not be treated as sole evidence for evaluating individual agents, customers, or business decisions.

Overall, we believe the main ethical contribution of this work is to encourage **more explicit, auditable, and failure-aware evaluation** of tool-using LLM systems in enterprise contexts where

silent planning errors could otherwise be difficult to detect.

References

Abubakari Alidu, Michele Ciavotta, and Flavio DePaoli. 2025. Prompt2dag: A modular methodology for llm-based data enrichment pipeline generation. *Preprint*, arXiv:2509.13487.

Anthropic. 2024. Claude 3.5 Haiku. https://www.anthropic.com/news/claude-3-5. Accessed: [Date of access].

Anthropic. 2025. Claude-sonnet-4. Large language model, accessed [date].

Pengfei Cao, Tianyi Men, Wencan Liu, Jingwen Zhang, Xuzhao Li, Xixun Lin, Dianbo Sui, Yanan Cao, Kang Liu, and Jun Zhao. 2025. Large language models for planning: A comprehensive and systematic survey. *Preprint*, arXiv:2505.19683.

Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

Jacob Cohen. 1968. Weighted kappa: Nominal scale agreement with provision for scaled disagreement or partial credit. *Psychological bulletin*, 70(4):213.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Preprint*, arXiv:2306.06070.

Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological Bulletin*.

Kai Goebel and Patrik Zips. 2025. Can llm-reasoning models replace classical planning? a benchmark study. *Preprint*, arXiv:2507.23589.

Amazon Artificial General Intelligence. 2024. The amazon nova family of models: Technical report and model card. *Amazon Technical Reports*.

Jungkoo Kang. 2025. Scaling llm planning: Nl2flow for parametric problem generation and rigorous evaluation. *Preprint*, arXiv:2507.02253.

J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *Biometrics*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Shicheng Liu, Jialiang Xu, Wesley Tjangnaka, Sina Semnani, Chen Yu, and Monica Lam. 2024. SUQL: Conversational search over structured and unstructured data with large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4535–4555, Mexico City, Mexico. Association for Computational Linguistics.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2023. Agentbench: Evaluating llms as agents. *Preprint*, arXiv:2308.03688.

Zixian Ma, Weikai Huang, Jieyu Zhang, Tanmay Gupta, and Ranjay Krishna. 2024. m&m's: A benchmark to evaluate tool-use for multi-step multi-modal tasks. *Preprint*, arXiv:2403.11085.

Meta. 2025. Llama 4 maverick. https://ai.meta.com/blog/llama-4-multimodal-intelligence/. Model released as a part of the Llama 4 family.

Kazuma Obata, Tatsuya Aoki, Takato Horii, Tadahiro Taniguchi, and Takayuki Nagai. 2024. Lip-llm: Integrating linear programming and dependency graph with large language models for multi-robot task planning. *Preprint*, arXiv:2410.21040.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, and 401 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

OpenAI. 2025. Gpt-4.1-mini. API Model. Available from OpenAI API https://openai.com/index/gpt-4-1/.

OpenAI. 2025a. GPT-5 System Card. https://openai.com/index/gpt-5-system-card/. Accessed: 2 January 2026.

OpenAI. 2025b. OpenAI o3-mini. https://openai.com/index/openai-o3-mini/. Announced January 31, 2025.

Vishal Pallagani, Bharath Chandra Muppasani, Kaushik Roy, Francesco Fabiano, Andrea Loreggia, Keerthiram Murugesan, Biplav Srivastava, Francesca Rossi, Lior Horesh, and Amit Sheth. 2024. On the prospects of incorporating large language models (llms) in automated planning and scheduling (aps). *Proceedings of the International Conference on Automated Planning and Scheduling*, 34:432–444.

Meta AI Research. 2024. Llama 3: An open-source framework for language modeling. Published via blog posts and official releases, as no specific public paper yet. Refer to the official announcements. The Llama 3.2 models, including the 1B and 3B instruct variants, are a subsequent release built upon the Llama 3 framework. The primary source for the

3.2 models is the AI at Meta blog post on Sept 25, 2024.

Gaurav Sahu, Abhay Puri, Juan Rodriguez, Amirhossein Abaskohi, Mohammad Chegini, Alexandre Drouin, Perouz Taslakian, Valentina Zantedeschi, Alexandre Lacoste, David Vazquez, Nicolas Chapados, Christopher Pal, Sai Rajeswar Mudumba, and Issam Hadj Laradji. 2025. Insightbench: Evaluating business analytics agents through multi-step insight generation. *Preprint*, arXiv:2407.06423.

Yewei Song, Xunzhu Tang, Cedric Lothritz, Saad Ezzini, Jacques Klein, Tegawendé F. Bissyandé, Andrey Boytsov, Ulrick Ble, and Anne Goujon. 2025. Callnavi, a challenge and empirical study on llm function calling and routing. *Preprint*, arXiv:2501.05255.

Marcus Tantakoun, Xiaodan Zhu, and Christian Muise. 2025. Llms as planning modelers: A survey for leveraging large language models to construct automated planning models. *Preprint*, arXiv:2503.18971.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Preprint*, arXiv:2206.10498.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *Preprint*, arXiv:2309.10691.

Ziting Wang, Shize Zhang, Haitao Yuan, Jinwei Zhu, Shifu Li, Wei Dong, and Gao Cong. 2025. Fdabench: A benchmark for data agents on analytical queries over heterogeneous data. *Preprint*, arXiv:2509.02473.

Duo Wu, Jinghe Wang, Yuan Meng, Yanning Zhang, Le Sun, and Zhi Wang. 2025a. Catp-llm: Empowering large language models for cost-aware tool planning. *Preprint*, arXiv:2411.16313.

Zirui Wu, Xiao Liu, Jiayi Li, Lingpeng Kong, and Yansong Feng. 2025b. Haste makes waste: Evaluating planning abilities of llms for efficient and feasible multitasking with time constraints between actions. *Preprint*, arXiv:2503.02238.

Jerrold H Zar. 2005. Spearman rank correlation. *Encyclopedia of Biostatistics*, 7.

A Tool Specifications and Domain Data

A.1 Tools and APIs

All tools are internally built; we expose schemas used in our experiments.

Common notation. Each tool call takes (i) an optional list of identifiers (e.g., `call_ids` or `interaction_ids`) passed positionally via a placeholder like "(k)" to denote step-k output, and (ii) a natural-language prompt. Placeholders MUST be used whenever a step depends on prior outputs.

A.1.1 T2S: Text-to-SQL over Snowflake

1. **Signature.** `T2S(call_ids: list, prompt: str) -> text`
2. **Function.** Generates SQL from `prompt`, optionally constraining results to `call_ids`; executes the SQL in Snowflake and returns a natural-language summary.
3. **Strengths.** Counts, trends, rollups over *structured* contact-center data.
4. **Limits.** Does not track *within-call* temporal dynamics (e.g., sentiment *shifts*); those require RAG.

A.1.2 RAG: Retrieval-Augmented Generation over Transcripts

1. **Signature.** `RAG(call_ids: list, prompt: str) -> text + table`
2. **Function.** Retrieves and analyzes customer–agent transcripts, returning (i) a textual answer and (ii) a *Data Insights* table with columns:
   - `Topic Name`, `Topic Description`
   - `Interaction Ids` (list)
   - `Interactions (%)` (sample-relative)
3. **Sampling.** For broad thematic prompts, RAG analyzes a sample (e.g., $\sim$200 interactions); percentages are with respect to the sample.
4. **Strengths.** Conversational content, phrasing, subjective judgments, *within-call* events (e.g., sentiment shifts).
5. **Limits.** Sample-based; ID extraction for downstream tools often requires an LLM reformat step.

A.1.3 LLM: Synthesis and Reformatting

1. **Signature.** `LLM(prompt: str) -> text`
2. **Function.** (i) merges multi-tool evidence; (ii) reformats outputs (e.g., extract `interaction_ids` from RAG tables); (iii) performs lightweight computations (set ops, joins of previously returned aggregates); (iv) synthesizes final answers.

LLM	Objective Queries			Subjective Queries		
	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)
gpt-4o	**50.53**	62.11	74.74	**41.18**	62.75	87.25
o3-mini	**47.37**	55.79	72.63	39.22	50.98	72.55
claude-3-7-sonnet	**33.68**	46.32	62.11	27.45	50.00	78.43
claude-sonnet-4	**32.63**	51.58	68.42	28.43	53.92	78.43
gpt-4o-mini	31.58	51.58	62.11	**32.35**	46.08	66.67
claude-3-5-haiku	**27.37**	44.21	56.84	22.55	47.06	67.65
nova-pro	**20.00**	36.84	55.79	16.67	48.04	67.65
llama3-70b-instruct	**18.95**	37.89	53.68	18.63	50.00	65.69
llama4-maverick-17b-instruct	18.95	32.63	48.42	**21.57**	40.20	59.80
nova-micro	17.89	30.53	49.47	**23.53**	40.20	56.86
gpt-4.1-nano	12.63	16.84	37.89	**21.57**	39.22	71.57
nova-lite	9.47	18.95	33.68	**17.65**	42.16	63.73
llama3-2-3b-instruct	1.05	3.16	9.47	**8.82**	17.65	24.51
llama3-2-1b-instruct	0.00	0.00	0.00	0.00	0.00	0.00
Grand Total (#)		**241**			**259**	

Table 5: Plan Generation quality comparison using prompts **with lineage** for **objective** and **subjective** queries, judged using the **one-shot evaluator** on test data. The highest score in the **Extremely Good, Very Good** bucket is highlighted in green for both query categories; blue indicates better performance on Objective queries, and magenta indicates better performance on Subjective queries.

LLM	Objective Queries			Subjective Queries		
	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)
o3-mini	**55.79**	67.37	80.00	**44.12**	64.71	80.39
gpt-4o	**43.16**	56.84	80.00	39.22	58.82	85.29
claude-3-5-haiku	**33.68**	49.47	69.47	27.45	46.08	75.49
gpt-4o-mini	**32.63**	44.21	58.95	31.37	59.80	83.33
claude-sonnet-4	27.37	41.05	65.26	**32.35**	57.84	82.35
claude-3-7-sonnet	**25.26**	43.16	68.42	14.71	35.29	69.61
nova-pro	**22.11**	40.00	52.63	17.65	43.14	65.69
llama3-70b-instruct	**22.11**	37.89	51.58	12.75	33.33	58.82
llama4-maverick-17b-instruct	**21.05**	38.95	57.89	20.59	42.16	66.67
nova-micro	**15.79**	33.68	46.32	15.69	40.20	60.78
gpt-4.1-nano	12.63	21.05	38.95	**19.61**	39.22	66.67
nova-lite	6.54	25.48	34.96	**20.49**	43.04	59.71
llama3-2-3b-instruct	3.16	5.26	14.74	**7.84**	17.65	38.24
llama3-2-1b-instruct	0.00	0.00	2.11	0.00	0.00	0.98
Grand Total (#)		**241**			**259**	

Table 6: Plan Generation quality comparison using prompts **without lineage** for **objective** and **subjective** queries, judged using the **one-shot evaluator** on test data. The highest score in the **Extremely Good, Very Good** bucket is highlighted in green for both query categories; blue indicates better performance on Objective queries, and magenta indicates better performance on Subjective queries.

A.2 Domain Data Fields in Snowflake

Interaction/Call metadata: channel (`call/chat/email`), timestamps, agent, team, customer/merchant/dasher tags.

Call drivers (*interaction drivers*): normalized reason categories (e.g., payment issues, delivery delays, missing items, account access, order status).

LLM	Simple Queries			Compound Queries		
	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)
o3-mini	**45.28**	51.89	69.81	40.66	54.95	75.82
gpt-4o	42.45	55.66	73.58	**49.45**	70.33	90.11
claude-3-7-sonnet	**41.51**	52.83	69.81	17.58	42.86	71.43
gpt-4o-mini	**36.79**	51.89	66.04	26.37	45.05	62.64
claude-sonnet-4	**33.02**	51.89	67.92	27.47	53.85	80.22
claude-3-5-haiku	**29.25**	49.06	62.26	19.78	41.76	62.64
llama4-maverick-17b-instruct	**28.30**	47.17	61.32	10.99	24.18	46.15
gpt-4.1-nano	**27.36**	35.85	60.38	5.49	19.78	49.45
nova-micro	**25.47**	38.68	54.72	15.38	31.87	51.65
nova-pro	**19.81**	46.23	61.32	16.48	38.46	62.64
llama3-70b-instruct	**19.81**	42.45	64.15	17.58	46.15	54.95
nova-lite	**16.98**	33.02	50.00	9.89	28.57	48.35
llama3-2-3b-instruct	**6.60**	10.38	17.92	3.30	10.99	16.48
llama3-2-1b-instruct	0.00	0.00	0.00	0.00	0.00	0.00
Grand Total (#)		**269**			**231**	

Table 7: Plan Generation quality comparison using prompts **with lineage** for **Simple** and **Compound** queries using the **one-shot evaluator** on test data. The highest score in the **Extremely Good, Very Good** bucket is highlighted in green for both query categories; blue indicates better performance on Simple queries, and magenta indicates better performance on Compound queries.

LLM	Simple Queries			Compound Queries		
	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)
o3-mini	**47.17**	63.21	75.47	**52.75**	69.23	85.71
gpt-4o	**44.34**	57.55	83.02	37.36	58.24	82.42
claude-sonnet-4	**38.68**	50.00	67.92	19.78	49.45	81.32
gpt-4o-mini	**36.79**	52.83	66.98	26.37	51.65	76.92
claude-3-5-haiku	**33.96**	50.94	66.98	26.37	43.96	79.12
llama4-maverick-17b-instruct	**29.25**	43.40	68.87	10.99	37.36	54.95
gpt-4.1-nano	**28.30**	43.40	61.32	2.20	15.38	43.96
nova-pro	**26.42**	46.23	61.32	12.09	36.26	57.14
llama3-70b-instruct	**23.58**	46.23	61.32	9.89	23.08	48.35
claude-3-7-sonnet	**21.70**	33.96	61.32	17.58	45.05	78.02
nova-micro	**19.81**	40.57	60.38	10.99	32.97	46.15
nova-lite	**16.66**	40.25	53.45	10.22	27.80	40.99
llama3-2-3b-instruct	**9.43**	16.98	33.02	1.10	5.49	19.78
llama3-2-1b-instruct	0.00	0.00	2.83	0.00	0.00	0.00
Grand Total (#)		**269**			**231**	

Table 8: Plan Generation quality comparison using prompts **without lineage** for **Simple** and **Compound** queries using the **one-shot evaluator** on test data. The highest score in the **Extremely Good, Very Good** bucket is highlighted in green for both query categories; blue indicates better performance on Simple queries, and magenta indicates better performance on Compound queries.

Moments: Events of interest extracted from agent-customer transcripts such as escalation, transfer, `sentiment_tag`, `abusive_interaction`, `issue_resolved`, `follow_up_required`.[3]

[3]Some moments (e.g., *shift* from negative→positive) are only detectable in transcripts and thus via RAG; Snowflake typically stores *call-level* sentiment tags.

LLM	[1, 4] steps in Best Possible Plan			[5, 15] steps in Best Possible Plan		
	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)
o3-mini	**52.48**	60.28	75.89	19.64	35.71	64.29
gpt-4o	**48.23**	62.41	78.72	**39.29**	62.50	87.50
gpt-4o-mini	**38.30**	54.61	68.09	16.07	33.93	55.36
claude-3-7-sonnet	**37.59**	54.61	72.34	12.50	32.14	66.07
claude-sonnet-4	**32.62**	52.48	70.21	25.00	53.57	82.14
claude-3-5-haiku	**28.37**	47.52	61.70	16.07	41.07	64.29
nova-micro	**26.24**	41.13	56.74	7.14	21.43	44.64
llama4-maverick-17b-instruct	**25.53**	44.68	60.28	7.14	16.07	39.29
llama3-70b-instruct	**24.11**	51.77	68.79	5.36	25.00	37.50
gpt-4.1-nano	**21.99**	32.62	58.16	5.36	17.86	48.21
nova-pro	**20.57**	46.10	63.83	12.50	33.93	57.14
nova-lite	**17.73**	34.04	51.77	3.57	23.21	42.86
llama3-2-3b-instruct	**7.09**	12.06	19.15	0.00	7.14	12.50
llama3-2-1b-instruct	0.00	0.00	0.00	0.00	0.00	0.00
Grand Total (#)		**358**			**142**	

Table 9: Plan Generation quality comparison using prompts **with lineage** for queries with **1 to 4 steps** and **5 to 15 steps** in the **Best Possible Plan** using the **one-shot evaluator** on test data. The highest score in the **Extremely Good, Very Good** bucket is highlighted in green for both the groups; blue indicates better performance on queries with [1,4] steps in the Best Possible Plan, and magenta indicates better performance on queries with [5,15] steps in the Best Possible Plan.

LLM	[1, 4] steps in Best Possible Plan			[5, 15] steps in Best Possible Plan		
	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)
o3-mini	**54.61**	71.63	80.85	**37.50**	51.79	78.57
gpt-4o	**43.26**	58.16	82.27	35.71	57.14	83.93
gpt-4o-mini	**34.75**	50.35	68.79	25.00	57.14	78.57
claude-sonnet-4	**34.75**	48.23	68.09	17.86	53.57	89.29
claude-3-5-haiku	**31.91**	51.06	70.92	26.79	39.29	76.79
llama4-maverick-17b-instruct	**25.53**	43.26	68.79	8.93	33.93	46.43
nova-pro	**23.40**	45.39	60.99	10.71	32.14	55.36
gpt-4.1-nano	**22.70**	37.59	57.45	0.00	12.50	42.86
claude-3-7-sonnet	**20.57**	36.17	63.83	17.86	46.43	82.14
llama3-70b-instruct	**20.57**	40.43	58.16	8.93	23.21	48.21
nova-micro	**16.31**	37.59	54.61	14.29	35.71	51.79
nova-lite	**15.60**	40.43	53.90	8.93	19.64	39.29
llama3-2-3b-instruct	**7.09**	12.77	28.37	1.79	8.93	23.21
llama3-2-1b-instruct	0.00	0.00	2.13	0.00	0.00	0.00
Grand Total (#)		**358**			**142**	

Table 10: Plan Generation quality comparison using prompts **without lineage** for queries with **1 to 4 steps** and **5 to 15 steps** in the **Best Possible Plan** using the **one-shot evaluator** on test data. The highest score in the **Extremely Good, Very Good** bucket is highlighted in green for both the groups; blue indicates better performance on queries with [1,4] steps in the Best Possible Plan, and magenta indicates better performance on queries with [5,15] steps in the Best Possible Plan.

QA metrics: overall score and category-level dimensions (e.g., Compliance & PII Security, Case Handling & Procedural Adherence, Timeliness, Professionalism, Empathy, Closing/Wrap-up, Communication Clarity), plus pass/fail flags for specific checks (e.g.,

LLM	Zero Hop (A+) Extr. good, very good (%)	One Hop (A+) Extr. good, very good (%)	Two Hop (A+) Extr. good, very good (%)	Three-Plus Hop (A+) Extr. good, very good (%)
claude-sonnet-4	**25.00**	31.40	**38.89**	17.24
llama4-maverick-17b-instruct	21.43	**26.74**	14.81	10.34
o3-mini	17.86	**55.81**	51.85	13.79
gpt-4o-mini	14.29	**45.35**	31.48	10.34
claude-3-7-sonnet	14.29	**41.86**	35.19	3.45
gpt-4o	10.71	51.16	**66.67**	**24.14**
nova-pro	7.14	**25.58**	16.67	10.34
llama3-70b-instruct	7.14	**25.58**	18.52	10.34
gpt-4.1-nano	7.14	**25.58**	16.67	3.45
nova-micro	0.00	**34.88**	14.81	10.34
nova-lite	0.00	**24.42**	7.41	6.90
llama3-2-3b-instruct	0.00	**10.47**	1.85	0.00
llama3-2-1b-instruct	0.00	0.00	0.00	0.00
claude-3-5-haiku	0.00	30.23	**33.33**	17.24
Grand Total (#)	**88**	**203**	**140**	**69**

Table 11: Plan Generation quality comparison using prompts **with lineage** for **Zero-Hop**, **One-Hop**, **Two-Hop** and **Three-Plus Hop** queries using the **one-shot evaluator** on test data. The highest score in the **Extremely Good, Very Good** bucket is highlighted in green for each group; The highest score across groups for each model is highlighted in blue.

"agent did not end call properly"). The category-level dimensions correspond to questions which are sourced from call evaluation forms, each designed to assess various aspects of agent performance during interactions.

A.3 Execution Constraints and Best Practices

- **DAG:** Dependencies D_k must form a DAG. Steps may execute in parallel when their dependencies are satisfied.
- **No redundant filtering:** If step k consumes IDs produced by step i, p_k should not restate filters used to produce those IDs (avoids unnecessary joins/timeouts).
- **RAG → ID extraction:** When a downstream step needs identifiers from a RAG table, insert an LLM step to extract/format the IDs (e.g., list of `interaction_ids`).
- **Tool capability guardrails:** Use RAG for *within-call* dynamics (e.g., sentiment shifts). Use T2S for structured aggregates (counts, trends, QA rollups).

B Full Example Lineage

Query. *"Analyze calls flagged as unresolved where the customer's sentiment transitioned from negative to positive within the transcript, and compare agent QA scores for resolution procedures versus professionalism."*

Initial → Final lineage (abbrev.).

1. $P^{(0)}$ *(weak):* Single-step T2S mixing unresolved status *and* within-call sentiment shift (unsupported) ⇒ timeout/low fidelity.
2. $P^{(1)}$*:* Split unresolved filtering (T2S) from sentiment shift (RAG), but missing ID extraction; downstream T2S cannot accept RAG table.
3. $P^{(2)}$*:* Insert LLM to extract `interaction_ids` from RAG's Data Insights; add QA rollups via T2S; missing final synthesis.
4. $P^{(3)}$ *(best):* 6-step plan as in §1 example: unresolved filter (T2S) → shift detection (RAG) → ID extraction (LLM) → QA rollups (T2S) → synthesis (LLM).

C Dataset Generation Details

C.1 Query Generation

We generate a pool of queries stratified across:

- **Subjectivity:** objective (counts, durations, resolution rates) vs. subjective (themes, phrasing, behaviors, sentiment).
- **Compoundness:** simple (single ask) vs. compound (multiple asks or comparisons).

This controls difficulty and ambiguity while matching realistic contact-center analytics needs.

LLM	Zero Hop (A+) Extr. good, very good (%)	One Hop (A+) Extr. good, very good (%)	Two Hop (A+) Extr. good, very good (%)	Three-Plus Hop (A+) Extr. good, very good (%)
nova-pro	**14.29**	23.26	**27.78**	0.00
llama4-maverick-17b-instruct	**14.29**	**26.74**	24.07	3.45
claude-sonnet-4	10.71	**39.53**	31.48	17.24
claude-3-7-sonnet	10.71	20.93	**25.93**	13.79
o3-mini	7.14	**62.79**	**62.96**	27.59
gpt-4o	7.14	47.67	**53.70**	**31.03**
nova-micro	3.57	17.44	**24.07**	6.90
nova-lite	3.57	**17.44**	14.81	10.34
llama3-70b-instruct	3.57	**24.42**	20.37	3.45
llama3-2-3b-instruct	3.57	5.81	**7.41**	3.45
gpt-4o-mini	3.57	**40.70**	35.19	27.59
gpt-4.1-nano	3.57	**29.07**	11.11	0.00
claude-3-5-haiku	3.57	34.88	**40.74**	24.14
llama3-2-1b-instruct	0.00	0.00	0.00	0.00
Grand Total (#)	**88**	**203**	**140**	**69**

Table 12: Plan Generation quality comparison using prompts **without lineage** for **Zero-Hop**, **One-Hop**, **Two-Hop** and **Three-Plus Hop** queries using the **one-shot evaluator** on test data. The highest score in the **Extremely Good, Very Good** bucket is highlighted in green for each group; The highest score across groups for each model is highlighted in blue.

LLM	With Lineage		Without Lineage	
	Objective Queries	Subjective Queries	Objective Queries	Subjective Queries
claude-3-7-sonnet	82.9	**85.58**	82.44	**84.3**
gpt-4o	81.94	**82.02**	**83.49**	83.44
claude-sonnet-4	**80.85**	80.09	76.1	**79.61**
llama4-maverick-17b-instruct	79.49	**84.21**	83.04	**83.5**
gpt-4o-mini	**78.64**	76.76	**84.35**	82.85
nova-pro	78.33	**79.73**	**82.78**	82.67
gpt-4.1-nano	**78.2**	77.59	**78.33**	75.52
llama3-3-70b-instruct	76.65	**81.06**	80.21	**82.09**
nova-micro	76.05	**77.28**	79.62	**83.28**
o3-mini	71.49	**72.64**	72.91	**75.83**
llama3-2-3b-instruct	70.5	**71.58**	75.14	**76.52**
claude-3-5-haiku	69.59	**72.39**	78.68	**83.63**
nova-lite	68.01	**71.81**	73.08	**77.51**
llama3-2-1b-instruct	20.18	**21.11**	**57.94**	55.53
Grand Total (#)	**241**	**259**	**241**	**259**

Table 13: Plan Generation quality comparison using prompts **with and without lineage** for **Objective and Subjective** queries, judged using the **Overall Score** of the **metric-wise evaluator** on test data (**500** queries). For the two groups - **With Lineage** and **Without Lineage** - blue cells denote better performance on Objective queries, while magenta cells denote better performance on Subjective queries.

C.2 Initial Plan Generation

We craft a two-part prompt:

1. **System prompt:** task definition; JSON schema for plans (step, depends_on); tool capabilities and guardrails (T2S/RAG/LLM).
2. **User prompt:** formatting constraints, placeholder rules, dependency DAG requirement, and few-shot exemplars.

We ablate tool-description verbosity (low/medium/high) and few-shot counts (1–15). The *medium* tool detail with 6–8 examples consistently avoids under-specification and prompt overload. The prompts used for this are provided in Table 27.

C.3 Iterative Evaluator→Optimizer Loop

The loop refines a plan without executing tools. It consists of:

Step-wise Evaluator. Inspects each step's tool, prompt, and dependencies; emits diagnostic tags

LLM	With Lineage		Without Lineage	
	Simple Queries	Compound Queries	Simple Queries	Compound Queries
claude-3-7-sonnet	**86.17**	82.99	**84.6**	81.51
llama4-maverick-17b-instruct	**85.01**	78.83	82.39	**82.53**
gpt-4o	**82.83**	79.59	**83.48**	81.73
nova-pro	**81.17**	77	**83.72**	81.4
llama3-3-70b-instruct	**80.36**	78.95	**82.42**	79.36
gpt-4o-mini	**78.64**	75.36	**83.38**	81.79
claude-sonnet-4	78.42	**81.58**	77.41	**78.6**
gpt-4.1-nano	**78.25**	75.68	**77.8**	73.77
nova-micro	**77.53**	77.18	**82.79**	81.16
claude-3-5-haiku	**73.27**	68.76	**83.02**	80.83
llama3-2-3b-instruct	**72.21**	68.35	**78.63**	72.33
o3-mini	71.05	**72.65**	73.64	**74.39**
nova-lite	**70.7**	70.49	74.22	**76.4**
llama3-2-1b-instruct	19.6	**21.13**	**56.93**	56.45
Grand Total (#)	**269**	**231**	**269**	**231**

Table 14: Plan Generation quality comparison using prompts **with and without lineage** for **Simple and Compound** queries, judged using the **Overall Score** of the **metric-wise evaluator** on test data (**500** queries). For the two groups - **With Lineage** and **Without Lineage** - blue cells denote better performance on Simple queries, while magenta cells denote better performance on Compound queries.

LLM	With Lineage				Without Lineage			
	Zero Hop	One Hop	Two Hop	Three Plus Hop	Zero Hop	One Hop	Two Hop	Three Plus Hop
nova-pro	**87.01**	80.34	77.64	77.69	83.16	**84.17**	81.82	83.2
llama3-3-70b-instruct	**85.4**	81.23	79.98	76.22	82.27	**82.56**	80.95	78.19
claude-3-7-sonnet	85.36	**86.53**	84.05	83.49	82.52	**84.81**	83.49	82.77
gpt-4o	**83.67**	83.41	79.84	79.39	72.55	**85**	83.14	81.03
llama4-maverick-17b-instruct	83.52	**85.1**	79.31	82.6	80.71	**84.43**	82.73	83.17
claude-sonnet-4	76.57	79.64	**81.68**	81.52	76.58	77.82	**79.79**	77.73
nova-micro	74.77	77.24	**77.97**	74.13	83.21	**83.32**	81.6	81.33
gpt-4o-mini	73.39	**78.25**	78.24	72.9	74.07	**84.35**	82.54	83.45
gpt-4.1-nano	69.9	**78.34**	77.4	75.94	**80.78**	78.79	73.48	73.28
nova-lite	67.94	70.47	**70.86**	70.11	74.94	73.91	**77.67**	76.22
o3-mini	66.23	72.33	**73.71**	70.57	76.14	**77.02**	73.34	73.63
llama3-2-3b-instruct	65.95	**72.15**	69.73	69.6	77.39	**78.31**	74.47	73.34
claude-3-5-haiku	63.45	**72.83**	69.08	71.49	74.17	82.29	**83.39**	80.21
llama3-2-1b-instruct	14.76	20.32	19.7	**22.3**	**61.93**	56.68	54.61	57.82
Grand Total (#)	**88**	**203**	**140**	**69**	**88**	**203**	**140**	**69**

Table 15: Plan Generation quality comparison using prompts **with and without lineage** grouped by **Number of Hops**, judged using the **Overall Score** of the **metric-wise evaluator** on test data (**500** queries). For the two groups - **With Lineage** and **Without Lineage** - blue cells denote the best performance across categories of **Number of Hops** for each model.

with justifications:

- **Incorrect tool** (capability mismatch).
- **Complex prompt** (needs decomposition).
- **Repeated detail** (repeating filters already implied by prior IDs).
- **Multi-tool prompt** (eligible for T2S and RAG; consider dual coverage).
- **Incorrect prompt** (format-related errors, incorrect or insufficient information in the prompt).
- **No change**.

Plan Optimizer. Consumes diagnostics and makes two passes:

- **Change 0** (local fix): apply minimal edits per tag (e.g., replace tool, simplify prompt, remove redundant filter).

LLM	With Lineage		Without Lineage	
	[1, 4] steps	[5, 15] steps	[1, 4] steps	[5, 15] steps
claude-3-7-sonnet	**86.07**	82.98	**84.29**	82.46
nova-pro	**83.16**	80.56	**82.85**	82.07
gpt-4o-mini	**82.92**	79.07	**83.78**	81.37
gpt-4o	**81.84**	76.03	**82.92**	78.1
llama4-maverick-17b-instruct	**80.63**	76.67	**83.7**	81.12
claude-sonnet-4	**79.78**	73.17	**83.82**	81.62
gpt-4.1-nano	79.3	**82.11**	77.7	**79.34**
llama3-3-70b-instruct	**78.19**	75.07	**82.43**	81.49
claude-3-5-haiku	**78.17**	76.02	**77.51**	73.41
nova-lite	**72.37**	69.79	**82.76**	81.08
llama3-2-3b-instruct	**72.12**	71.72	**74.44**	74.3
o3-mini	**72.02**	67.53	**77.38**	72.65
nova-micro	70.12	**71.34**	74.37	**76.73**
llama3-2-1b-instruct	19.49	**21.43**	**58.76**	52.68
Grand Total (#)	**358**	**142**	**358**	**142**

Table 16: Plan Generation quality comparison using prompts **with and without lineage** grouped by **Number of steps in the Best Possible Plan**, judged using the **Overall Score** of the **metric-wise evaluator** on test data (**500** queries). For the two groups - **With Lineage** and **Without Lineage** - blue cells denote better performance on queries with [1, 4] steps in the best possible plan, while magenta cells denote better performance on queries with [5, 15] steps in the best possible plan.

Tag	Simple Queries				Compound Queries			
	Pre-Loop		Post-Loop		Pre-Loop		Post-Loop	
	#	%	#	%	#	%	#	%
Extremely Bad	8	2.97	13	4.83	20	8.66	10	4.33
Very Bad	71	26.39	58	21.56	56	24.24	51	22.08
Bad	46	17.10	51	18.96	61	26.41	71	30.74
Acceptable	36	13.38	23	8.55	30	12.99	25	10.82
Good	63	**23.42**	46	17.10	41	**17.75**	38	16.45
Very Good	30	11.15	48	**17.84**	15	6.49	25	**10.82**
Extremely Good	15	5.58	30	**11.15**	8	3.46	11	**4.76**
Grand Total	**269**	**100.00**	**269**	**100.00**	**231**	**100.00**	**231**	**100.00**

Table 17: Effectiveness of the **iterative Evaluator→Optimizer loop** judged using **one-shot evaluator** broken down by simple and compound queries on test data. Higher values between **Pre-Loop** and **Post-Loop** in the **Good, Very Good, Extremely Good** buckets are highlighted in green for each group.

- **Change 1** (coherence fix): repair global consistency (dependencies, placeholders), add missing split steps, merge redundancies.

Each revision is stored, forming the *plan lineage* for the query.

C.4 Plan Refinement via Evaluator→Optimizer Loop

Overview. Given an initial plan $P^{(0)}$, we iteratively improve its structure and correctness without executing tools. At each iteration, a *Step-wise Evaluator* diagnoses issues at the step level; a *Plan Optimizer* then applies edits and emits a revised plan, which is stored in the *lineage*.

Step-wise Evaluator. Input: a single step (tool + prompt) and its declared dependencies. Output: a *set* of tags plus brief rationales. Supported tags:

- INCORRECTTOOL — tool capability and prompt intent mismatch.
- COMPLEXPROMPT — prompt needs decomposition into simpler atomic steps.
- REPEATEDDETAIL — repeated filters (e.g., re-filtering on a criterion already used to create the call/interaction ID set).
- MULTITOOLPROMPT — task can (or should) be covered by both T2S and RAG for completeness.
- INCORRECTPROMPT — Prompt has discrepancies such as format-related issues (parentheses, placeholders, or dependency notation errors) and insufficient or incorrect information.

Tag	Objective Queries				Subjective Queries			
	Pre-Loop		Post-Loop		Pre-Loop		Post-Loop	
	#	%	#	%	#	%	#	%
Extremely Bad	13	5.39	10	4.15	15	5.79	13	5.02
Very Bad	86	35.68	66	27.39	41	15.83	43	16.60
Bad	58	24.07	66	27.39	48	18.53	56	21.62
Acceptable	23	9.54	18	7.47	43	16.60	30	11.58
Good	46	**19.09**	33	13.69	58	**22.39**	51	19.69
Very Good	5	2.07	25	**10.37**	41	15.83	48	**18.53**
Extremely Good	10	4.15	23	**9.54**	13	5.02	18	**6.95**
Grand Total	**259**	**100.00**	**259**	**100.00**	**241**	**100.00**	**241**	**100.00**

Table 18: Effectiveness of the **iterative Evaluator→Optimizer loop** judged using **one-shot evaluator** broken down by objective and subjective queries on test data. Higher values between **Pre-Loop** and **Post-Loop** in the **Good, Very Good, Extremely Good** buckets are highlighted in green for each group.

Tag	Zero-Hop				One-Hop			
	Pre-Loop		Post-Loop		Pre-Loop		Post-Loop	
	#	%	#	%	#	%	#	%
Extremely Bad	3	3.41	0	0.00	8	3.94	10	4.93
Very Bad	55	62.50	45	51.14	33	16.26	28	13.79
Bad	18	20.45	28	31.82	28	13.79	36	17.73
Acceptable	5	5.68	0	0.00	30	14.78	20	9.85
Good	0	0.00	0	0.00	66	**32.51**	51	25.12
Very Good	0	0.00	8	**9.09**	28	13.79	38	**18.72**
Extremely Good	7	7.95	7	7.95	10	4.93	20	**9.85**
Grand Total	**88**	**100.00**	**88**	**100.00**	**203**	**100.00**	**203**	**100.00**

Table 19: Effectiveness of the **iterative Evaluator→Optimizer loop** judged using **one-shot evaluator** broken down by number of hops (0-hops and 1-hop) on test data. Higher values between **Pre-Loop** and **Post-Loop** in the **Good, Very Good, Extremely Good** buckets are highlighted in green for each group.

- NoChange — the step is acceptable as-is.

Notably, the evaluator *does not* receive the original query as input (empirically improving consistency).

Plan Optimizer. Input: current plan P, step index i, diagnostic tags and rationales. Output: revised plan P'. The optimizer performs:

1. **Change 0 (local repair):** swap misaligned tools, simplify/decompose complex prompts, remove redundant filters, fix format.

2. **Change 1 (coherence repair):** rewire dependencies/placeholders for a valid DAG, split/merge steps if required, ensure that outputs used downstream have a unique producer and are referenced exactly once where intended.

Each time $P' \neq P$, we append P' to the lineage.

Control flow and guarantees. Algorithm 1 implements a bounded-pass scan: we either advance to the next step when length is unchanged or recheck the same index when structure changes. The outer guard ($pass < M$) ensures termination; see Lemma 1. The loop improves plans textually and deterministically modulo LLM sampling settings (we fix temperature and seeds during curation).

Tag	Two-Hops				Three-Plus-Hops			
	Pre-Loop		Post-Loop		Pre-Loop		Post-Loop	
	#	%	#	%	#	%	#	%
Extremely Bad	15	10.71	5	3.57	3	4.35	8	11.59
Very Bad	13	9.29	20	14.29	26	37.68	15	21.74
Bad	43	30.71	38	27.14	17	24.64	20	28.99
Acceptable	20	14.29	23	16.43	10	14.49	5	7.25
Good	31	**22.14**	25	17.86	8	11.59	8	11.59
Very Good	13	9.29	18	**12.86**	5	7.25	10	**14.49**
Extremely Good	5	3.57	11	**7.86**	0	0.00	3	**4.35**
Grand Total	**140**	**100.00**	**140**	**100.00**	**69**	**100.00**	**69**	**100.00**

Table 20: Effectiveness of the **iterative Evaluator→Optimizer loop** judged using **one-shot evaluator** broken down by number of hops (2-hops and 3-plus-hops) on test data. Higher values between **Pre-Loop** and **Post-Loop** in the **Good, Very Good, Extremely Good** buckets are highlighted in green for each group.

Metric	Reference-Free		Reference-Based	
	Single	Deconstructed	Single	Deconstructed
	%	%	%	%
Dependency	62.74	96.00	67.81	**97.3**
Format	39.21	93.46	53.85	**94.02**
Step Executability	58.36	72.55	60.24	**79.43**
Query Adherence	39.84	75.82	47.52	**85.42**
Redundancy	53.00	85.63	56.33	**88.85**
Tool Prompt Alignment	49.02	65.36	52.94	**81.29**
Tool Usage Completeness	11.76	60.13	42.79	**94.12**

Table 21: Evaluation of **Metric-Wise Evaluator**: Triplet Ranking Agreement (Relaxed) Between **Metric-Wise Evaluator** and **Human Annotations** on the Validation Set (N=80): Percent of Correct Inequalities per Metric. The best-performing setting is highlighted for every metric. The top-performing setting is highlighted for each metric: values $\geq 90\%$ in blue, $\geq 80\%$ in magenta, and $< 80\%$ in red.

Tag	# Annotator Assigned	# Evaluator Assigned	# Overlap	Precision	Recall	F1-Score
Extremely Bad	9	9	9	1.00	1.00	**1.00**
Very Bad	7	6	6	1.00	0.86	**0.92**
Bad	16	15	14	0.93	0.88	**0.90**
Acceptable	19	17	16	0.94	0.84	**0.89**
Good	9	9	9	1.00	1.00	**1.00**
Very Good	12	14	11	0.79	0.92	**0.85**
Extremely Good	8	10	8	0.80	1.00	**0.89**
Overall (Macro Average)	80	80	73	0.92	0.93	**0.92**

Table 22: Evaluation of **One-Shot Evaluator**: Agreement in Assigning Tag Between **One-Shot Evaluator** and **Human Annotations** on the Validation Set (N=80). F1-Scores $\geq 90\%$ in blue, and $\geq 80\%$ in magenta.

Tag	# Annotator Assigned	# Evaluator Assigned	# Overlap	Precision	Recall	F1-Score
Extremely Bad	9	11	9	0.82	1.00	**0.90**
Very Bad	7	8	5	0.62	0.71	0.67
Bad	16	15	15	1.00	0.94	**0.97**
Acceptable	19	16	16	1.00	0.84	**0.91**
Good	9	8	8	1.00	0.89	**0.94**
Very Good	12	13	12	0.92	1.00	**0.96**
Extremely Good	8	9	7	0.78	0.88	**0.82**
Overall (Macro Average)	80	80	72	0.88	0.89	**0.88**

Table 23: Evaluation of **GPT-5** as **One-Shot Evaluator**: Agreement in Assigning Tag Between **One-Shot Evaluator** and **Human Annotations** on the Validation Set (N=80). F1-Scores $\geq 90\%$ in blue, and $\geq 80\%$ in magenta.

Metric	ρ	$p - value$
Total metric-wise score	0.60	0.023
Oneshot overall quality	0.52	0.059
Dependency	0.84	0.0002
Format	0.58	0.029
Redundancy	0.55	0.043
Step Executability	0.54	0.047
Tool Usage Completeness	**0.35**	**0.217**
Query Adherence	0.79	0.0007
Tool–Prompt Alignment	0.72	0.0035

Table 24: Spearman rank correlations between LLM rankings under **Claude-Sonnet-4** and **GPT-5** as Judges on a 50-query test subset (14 planners, prompt type = *Without Lineage*). $\rho < 0.5$, and $p - value > 0.1$ are highlighted in red.

Tag	# Annotator Assigned	# Evaluator Assigned	# Overlap	Precision	Recall	F1-Score
No Change	66	70	51	0.77	0.73	**0.75**
Incorrect Tool	66	62	55	0.83	0.89	**0.86**
Incorrect Prompt	68	66	61	0.9	0.92	**0.91**
Complex Prompt	70	63	53	0.76	0.84	**0.8**
Repeated Detail	74	71	66	0.89	0.93	**0.91**
Multi-tool Prompt	56	68	50	0.89	0.74	**0.81**
Overall (Macro Average)	400	400	336	0.84	0.84	**0.84**

Table 25: Evaluation of **Step-Wise Evaluator**: Agreement in Assigning Tag Between **Step-Wise Evaluator** and **Human Annotations** on the Validation Set (N=400). F1-Scores $\geq 90\%$ in blue, $\geq 80\%$ in magenta and $< 80\%$ in red.

Tag	#	%
Extremely Bad	2	1.25
Very Bad	8	5
Bad	14	8.75
Acceptable	17	10.63
Good	35	**21.88**
Very Good	39	**24.38**
Extremely Good	45	**28.13**
Total	**160**	**100**

Table 26: Evaluation of **Plan Optimizer**: Similarity Between Plans Generated By **Plan Optimizer** and **Human Annotations** (considered as reference) on the Validation Set (N=160) Computed Based on **One-Shot Evaluator**. The proportion of plans generated by the Plan Optimizer falling in the top 3 buckets - {**Extremely Good, Very Good, Good**} are highlighted in blue.

Module	Prompt
Query Generation	**SYSTEM_PROMPT:** You are an expert query generator. The queries you generate should mimick real-life user queries related to a call center with agents. The queries should aim to gain insight into the call center's functioning and performance. Here are some reference examples. Generate 10 new queries. {QUERY_REFERENCE_EXAMPLES}
Plan Generation (Without Lineage)	**SYSTEM_PROMPT:** You are a high-level planner agent designed to orchestrate step-by-step solutions to business queries related to call center operations. Your role is to: - Analyze the given query and break it down into a coherent, **step-by-step plan**. - Ensure that **all steps rely solely on the tools provided below** — do not assume access to any external data, tools, or knowledge beyond what is defined. - Each step of the output plan should be simple and executable by specialized tool-aware agents. You have access to the following tools: {TOOL_DESCRIPTIONS} **MAIN_PROMPT:** Task: 1. You will be provided with a query to answer. 2. Generate a high-level, step-by-step plan to solve it. 3. Use only the tools available to you (described above). 4. Design the steps so that the plan is logically valid, realistically achievable, and contextually appropriate. 5. Clearly define dependencies between steps where intermediate results are needed. 6. Output each plan in the following format: <example number>: Query: <provided query> Plan: { <step number>: {"step": <tool name>("<input instruction to the tool>"), "depends_on": [<step numbers that this step depends on>]} .... } Guidelines: - Be structured: respect execution order and dependencies - Be concise: focus each step on a specific actionable instruction - Return ONLY the plan Reference: Below are some real examples of business queries and their corresponding multi-step plans along with thought process behind the plans: {EXAMPLE_QUERY_PLANS_EXPLANATIONS} The input query to answer is: {QUERY}
Plan Generation (With Lineage)	**SYSTEM_PROMPT:** You are a high-level planner agent designed to orchestrate step-by-step solutions to business queries related to call center operations. Your role is to: - Analyze the given query and break it down into a coherent, **step-by-step plan**. - Ensure that **all steps rely solely on the tools provided below** do not assume access to any external data, tools, or knowledge beyond what is defined. - Each step of the output plan should be simple and executable by specialized tool-aware agents. You have access to the following tools: {TOOL_DESCRIPTIONS} **MAIN_PROMPT:** Task: 1. You will be provided with a query to answer. 2. Generate a high-level, step-by-step plan to solve it. 3. Use only the tools available to you (described above). 4. Design the steps so that the plan is logically valid, realistically achievable, and contextually appropriate. 5. Clearly define dependencies between steps where intermediate results are needed. 6. Output each plan in the following format: <example number>: Query: <provided query> Plan: { <step number>: {"step": <tool name>("<input instruction to the tool>"), "depends_on": [<step numbers that this step depends on>]} .... } Guidelines: - Be structured: respect execution order and dependencies - For each step k, the depends_on list may only reference step indices in 1, . . . , k-1, and the overall dependency graph must remain acyclic (no cycles). - Be concise: focus each step on a specific actionable instruction - Return ONLY the plan Reference: Below are examples of business queries paired with sequences of plans. Each sequence begins with a suboptimal plan and gradually improves through multiple iterations until reaching the best possible plan for the query. At each step, the plan is refined to better align with tool usage, reduce unnecessary complexity, and improve overall clarity. Use these plan lineages to identify common mistakes made by planners and to learn strategies for avoiding them. Given a query, generate an initial plan, then iteratively refine it, and return the full sequence of improved plans. {EXAMPLE_QUERY_PLANS_LINEAGES} The input query to answer is: {QUERY}

Table 27: Prompts Used For **Generation** of **Queries** and **One-Shot Plans**

Prompt
You are a **Step Evaluator Agent** responsible for carefully analyzing individual steps within a multi-step plan designed to answer a user query. Your mission is to evaluate each step against its dependent inputs and explain if there are any issues with the step, using the **standardized error types and format**. You need to carefully determine whether the step has errors and justify your evaluation based on the **reference examples** and the provided **tool capabilities**. ### **Your Task:** For a given **step in a plan**, you will evaluate and output the results as follows: 1. Analyze the step and its **dependent steps** to identify potential issues under the following **error types**: - **INCORRECT PROMPT** - **COMPLEX PROMPT** - **INCORRECT TOOL** - **REPEATED DETAIL** - **MULTI-TOOL PROMPT** - **NO CHANGE** 2. Follow this **order of judgment** for your evaluations: - Evaluate if the step has any of the following critical issues: **INCORRECT PROMPT**, **COMPLEX PROMPT**, or **INCORRECT TOOL**, comparing with the dependent step(s) if necessary. - If no critical issues are identified, assess the step for **REPEATED DETAILS** by comparing it to its dependent steps. - Check if the task in the step could also be performed by a different tool, and if it can then flag it as **MULTI-TOOL PROMPT**. - If none of the above issues are detected, then and only then, return **NO CHANGE**. 3. When identifying an **error**, include both: - A **clear, concise reasoning** for why that error applies. - The **appropriate error tag** corresponding to the error type. 4. **Output Format**: Use the following standardized output format for any identified issues: " <STEP NUMBER> 1. <REASONING BEHIND THE TYPE OF ERROR>: <ERROR TYPE> 2. <REASONING BEHIND THE TYPE OF ERROR>: <ERROR TYPE> ... " 5. Structure Of The Steps: <step_number>: {{"step": <Tool_Name>((placeholders), "<prompt>"), "depends_on": [dependent_steps]}} (when tool is RAG or T2S, placeholder is outside the prompt) <step_number>: {{"step": <Tool_Name>("<prompt>"), "depends_on": [<dependent_steps>]}} (when tool is LLM, placeholder is inside the prompt) ### **When Evaluating a Point, Some Key Points to Consider:** - **Step Structure**: Ensure the step matches the expected JSON format and adheres to placeholders (e.g., make sure any referenced dependent steps are properly reflected both in the ‘depends_on‘ array and the tool’s prompt using placeholders like ‘(1)‘ or ‘(2)‘). - **Dependent Steps**: Evaluate the dependent steps in relation to the current step. Any details already covered in the dependent steps should not be redundantly included in the current step’s prompt. - **Tool Assignment**: Each step must be handled by the appropriate tool. Use the **tool capabilities** section to determine whether the assigned tool is suitable for the task. A mismatch in tool functionality would lead to an **INCORRECT TOOL** error. ### **Error Type Definitions:** Here is the list of error types that you must use in your evaluations, along with their specific definitions: 1. **INCORRECT PROMPT**: - This error applies when the prompt given to the tool is incomplete or misaligned with the tool’s capabilities. This issue can lead to incorrect or incomplete outputs. - For example, if the query asks for calls, and the prompt to T2S extracts interaction_ids of all interactions, then that is an incorrect prompt that needs to be changed. If the placeholders are missing from an LLM step, that is an incorrect prompt and needs to be changed to include the placeholders. 2. **COMPLEX PROMPT**: - This error applies when the prompt is too complex to be executed by the tool in a single step. - Such prompts need to be broken down into simpler sub-steps. - This can happend when T2S is being asked to extract more than 1 thing at a time or when RAG is being asked to answer multiple questions at a time. If it can be sensibly broken down into simpler steps, it should be. 3. **INCORRECT TOOL**: - This error applies when the wrong tool is assigned to a particular task. - Look carefully at the tool description provided below, it explains in detail what kind of information can be answered by T2S and which can be answered by RAG. 4. **REPEATED DETAIL**: - This error applies if the step contains unnecessary redundancies, such as duplicating details already handled by a dependent step. 5. **MULTI-TOOL PROMPT**: - This error applies when the task assigned to a particular tool can also be accomplished with another available tool as well (e.g., if both T2S and RAG could perform the task). 6. **NO CHANGE**: - This result is valid when the step is completely correct, and none of the above points apply to it. — ### **Evaluation Order:** When evaluating, follow this strict judgment sequence: 1. Look for **critical issues** in the step: - Does the step contain an **INCORRECT PROMPT**, a **COMPLEX PROMPT**, or an **INCORRECT TOOL** error? 2. Next, assess if there are **REPEATED DETAILS** by comparing the step to its dependent steps. 3. Check if the task described in the step can also be performed by another tool. If so, flag it as a **MULTI-TOOL PROMPT**. 4. If none of the above apply, conclude your evaluation with **NO CHANGE**. — ### **Tools Available:** {Snowflake_tools_descriptions.TOOL_DESCRIPTIONS_ULTRASHORT} — ### **Reference Examples:** {Evaluator_creator.REFERENCE_EXAMPLES} ***IMPORTANT REMINDER*** - Order of dependencies and placeholders does not matter. Dependencies are there to know which steps this step is dependent on. Placeholders in a prompt so that when that prompt is finally given to the respective tool, we know where to put the output of the dependent step into the prompt.

Table 28: System Prompt Used For **Step-Wise Evaluator.**

Prompt
ROLE: You are an **Expert Plan Optimizer** tasked with refining and updating multi-step plans designed to answer complex queries related to call centers. You will be provided with a plan, along with the output of a step-evaluator. This output contains a detailed analysis of a particular step of the plan. Based on the output of the step-evaluator, you will make changes to the plan and return the updated plan. — ### **GOAL:** Produce a **revised multi-step plan** by: 1. Analyzing the output of the stepwise plan evaluator. 2. Based on the output of the step-evaluator, make changes to that particular step (*CHANGE 0*). 3. If necessary, make changes to the latter steps of the plan in order to maintain overall logical coherence (*CHANGE 1*). — ### **INPUTS:** 1. **Original Query:** - The query from the user that the plan is attempting to answer. 2. **Original Plan:** - A JSON-formatted textual representation of the original multi-step plan. - Each step of the plan should strictly adhere to the following format: <step_number>: {{"step": <tool_name>("<prompt>"), "depends_on": [<dependent_step_numbers>]}} **Key Details About Step Format:** - It must be JSON parsable, ensuring that details like double quotes, braces, commas, etc., are accurate. - Referenced steps by number (e.g., "depends_on": [1, 2]) must be reflected as placeholders (e.g., "(1)", "(2)") in the corresponding '<prompt>', especially in LLM steps. 3. **Step Evaluator Output:** - Feedback related to a specific step in the plan. - Includes: - **Step Number:** The step under scrutiny. - **Feedback:** Specific insights into potential issues such as incorrect prompt/tool, overly complex prompt, repeated details, etc. - **Error Type:** The categorization of the issue (e.g., INCORRECT PROMPT, INCORRECT TOOL, COMPLEX PROMPT, etc.). — ### **EXPECTED OUTPUT:** 1. **CHANGE 0:** - Update the problematic step directly based on the output of the step evaluator. Changes may include: - Prompt refinement. - Prompt breakdown resulting in the creation of new steps. - Tool reassignment. - Correction of repeated details. - Addition of new step based on "multi-tool prompt" output, if it is not already present. - Clearly and concisely explain what you changed and why. 2. **CHANGE 1:** - Address how the rest of the plan may need adjustments to ensure logical coherence based on **CHANGE 0**. - Must **NOT** include any plan optimization or improvement, only corrections to ensure that previous change does not break the entire plan. - Changes may include: - Updating indices of steps based on addition or removal of steps. - Updating dependencies/placeholders for downstream steps. - Adding additional steps for aggregation or reasoning. - Removing now-redundant or irrelevant steps. - Provide a detailed reasoning for any adjustments made. 3. **Revised Plan:** - A JSON-formatted version of the new, corrected plan. - Ensure correct syntax, adherence to format, and logical coherence throughout the revised plan. — ### **ERROR TYPES AND HOW TO HANDLE THEM:** Here is a comprehensive list of potential issues the plan optimizer might encounter and the strategies to address them. {STEP_OPTIMIZER_ERROR_CATEGORIES_DESCRIBED} — ### **FLOW:** Explain your analysis and actions in a two-stage process: #### **CHANGE 0:** - This stage addresses the feedback provided by the step evaluator for the particular step under consideration. - Describe the modifications made to this step and the rationale behind them explicitly. #### **CHANGE 1:** - This stage ensures coherence of the plan after implementing CHANGE 0. - Adjust dependencies, add/delete/update steps as necessary to ensure the plan works seamlessly after the initial change. - Your job is not to perform optimizations on the plan to improve it. It is simply to make any changes necessary to ensure that the rest of the plan isn't wrong because of CHANGE 0. - An example of what you SHOULD do: If a step has been removed and that step was referenced in another step's dependency, then your job is to correct all the step indices based on the removed step, and then to update the future step's dependencies and placeholders to account for this change. - An example of what you SHOULD NOT do: If a step was detected to have repeated detail and CHANGE 1 removed that repeated detail, then it is NOT your job to find similar repeated details in future steps and change them as well. - This is a VERY important distinction and should be remembered at all times. — ### **FORMAT OF INPUT AND OUTPUT:** INPUT FORMAT: - Query: <query> - Plan: <plan> - Step Evaluator Output: <step_evaluator_output> OUTPUT FORMAT: "CHANGE 0": <explanation of direct changes based on the step evaluator output> "CHANGE 1": <explanation of changes to the rest of the plan to ensure logical coherence> "NEW PLAN STARTS": <revised plan> — ### **TOOLS:** {Snowflake_tools_descriptions.TOOL_DESCRIPTIONS_ULTRASHORT} — ### **REMINDER:** - Follow the correct step format strictly. - Ensure all syntax, placeholders, and dependencies are accurate. - Changes must make the plan more logically connected and operationally sound with the tools provided. - Make sure that CHANGE 1 leads to correct and sequential indices in the new plan.

Table 29: System Prompt Used For **Plan Optimizer**

Module	Prompt
Dependency (Metric-wise Evaluator)	**You are an expert plan evaluation agent** You will be given a plan and your task is to meticulously evaluate its dependencies and assign it a score. * Here is an example of a plan: { 1: {"step": T2S([], "Fetch call_ids of calls related to payment adjustment inquiries."), "depends_on": []}, 2: {"step": RAG((1), "What patterns of customer sentiment emerge in these calls?"), "depends_on": [1]} } Here are the things to keep in mind: * **The only part of this plan that is relevant for you is the list followed by "depends_on" in each step. That list contains the steps that the current step depends on. * ** Your job is to find if for any step, that list of dependent steps is wrong which can only be in 1 of 2 cases. Either there is some dependent step that is missing from the "depends_on" list or there is some step in the "depends_on" list that shouldn't be there. * **Return the total steps - total number of steps with incorrect dependencies.** Input Format: You will be simply provided with a plan. Output Format: You must provide a detailed explanation of your analysis related to that point, and then give the final score as per your verdict. Follow the below provided format: <explanation> | <score> | Here are some reference examples: {DEPENDENCY_REFERENCE_EXAMPLES}
Format (Metric-wise Evaluator)	**You are an expert plan evaluation agent** You will be given a plan and your task is to meticulously evaluate its format and assign it a score. * Here is the correct format: Plan: {{ <step number>: {{"step/query": <tool name>("<input instruction to the tool>"), "depends_on": [<step numbers that this step depends on>]}} .... }} Here are the things to keep in mind: * **Return the total steps - total number of format violations.** * **Make sure that the plan is json parsable. If that fails due to thing such as quadruple quotes instead of double quotes, mismatched paranthesis or some similar reason, then that is a violation.** * **Make sure that every step, that has a dependency on another step, has a valid placeholder. The placeholder is the dependent step number, surrounded by paranthesis, within the prompt. For example in step 2 of the above correct plan, "RAG((1), "Wha...)", here "(1)" is the placeholder. Similarly a placeholder MUST exist in every step with a dependency, ESPECIALLY in the steps with LLM. Each step with an absent placholder or an incorrect placeholder (e.g. (1, 2) instead of (1) and (2) etc.) is a format violation.** * **If a step, for e.g an LLM step has a mention of the query, for e.g. "Answer the question/query based on ...", then it MUST have a placeholder for the query (e.g. (query)) otherwise that is a format violation.** Input Format: You will be simply provided with a plan. Output Format: You must provide a detailed explanation of your analysis related to that point, and then give the final score as per your verdict. Follow the below provided format: <explaination> | <score> | Here are some reference examples: {FORMAT_REFERENCE_EXAMPLES}
Redundancy (Metric-wise Evaluator)	**You are an expert plan evaluation agent** You will be given a plan and your task is to meticulously evaluate its redundancy and assign it a score. * Here is the format of a plan: Plan: {{ <step number>: {{"step/query": <tool name>("<input instruction/prompt to the tool>"), "depends_on": [<step numbers that this step depends on>]}} .... }} There are only two types of redundancies that you need to look for: 1. **T2S Step Redundancy:** - In a T2S step, if the instruction/prompt provided contains some information that is already accounted for in its dependent step, then that is a redundancy. - The reason being that for T2S i.e. Snowflake structured database specifically, any additional detail in the prompt can lead to additional table joins or other things that are not needed. - Due to this reason, this step redundancy ONLY APPLIES TO T2S AND NOT TO RAG OR LLM STEPS. - Phrases like "in these calls" or "in these interactions" in a T2S step is NOT considered to be redundant. - Example for this will be provided in the reference examples. 2. **Ignored Step Redundancy:** - The final step of a plan is the overall output of the plan. - If there is a step or steps that do not contribute to the final step, then those steps are redundant. - This can be found by looking at the "depends_on" list of each step. - Example for this will be provided in the reference examples. Return the total steps - total number of redundancies/redundant steps. Input Format: You will be provided with a plan. Output Format: You must provide a detailed explanation of your analysis related to that point, and then give the final score as per your verdict. Follow the below provided format: <explanation> | <score> | Here are some reference examples: {REDUNDANCY_REFERENCE_EXAMPLES}

Table 30: System Prompts Used in **Metric-Wise Evaluator** (Dependency, Format and Redundancy)

Module	Prompt
Step-Executability (Metric-wise Evaluator)	**You are an expert plan evaluation agent** You will be given a plan along with the best possible plan and your task is to meticulously evaluate its step-executability and assign it a score. * Here is the format of a plan: Plan: { <step number>: {"step/query": <tool name>("<input instruction/prompt to the tool>"), "depends_on": [<step numbers that this step depends on>]} .... } Here are the things to keep in mind: * The only thing that you need to look at is the tool and the corresponding instruction/prompt in each step. * Assume that everything that the prompt is asking from the tool is perfectly within the tool's capabilities. * Under this assumption, judge whether the prompt is too complex to be executed by the tool in 1 go and needs to be broken down into multiple steps. * There are 2 tools that you need to check, RAG (which performs Retrieval-Augmented Generation on some unstructured data) and T2S (which performs Text-to-SQL on some structured data). * For RAG: The prompt should only ask the tool to do one thing at a time. Examples provided in the reference examples. * For T2S: For a T2S step, the number of trigger words/trigger phrases present in the prompt is directly proportional to the complexity of the step. Here is a list of trigger words to look for. Don't look exactly for these trigger words, but look for similar looking phrases as well: ** For call drivers ** {call_drivers_op} ** For moments ** {moments_op} ** For qa ** {qa_op} Based on extracted data, T2S has the ability to perform simple mathematical operations such as finding mean etc. If a T2S step contains more than 3 trigger words/phrases, then it is not executable in 1 go. REMEMBER: "fetch interaction_id" or "fetch call_id" is NOT considered as a trigger phrase Return the total steps - total number of steps that are not executable in 1 go. Input Format: You will be provided with 2 plans, the plan in question and the best possible plan. Output Format: You must provide a detailed explanation of your analysis related to that point, and then give the final score as per your verdict. Follow the below provided format: <explanation> | <score> | Here are some reference examples: {STEP_ATOMICITY_REFERENCE_EXAMPLES}
Query Adherence (Metric-wise Evaluator)	**You are an expert plan evaluation agent** You will be given a query, the corresponding plan and the best possible plan, and your task is to meticulously evaluate its query adherence and assign it a score. * Here is the format of a plan: Plan: { <step number>: {"step/query": "TOOL"("<input instruction to the tool>"), "depends_on": [<step numbers that this step depends on>]} .... } Here are the things to keep in mind: * Do not worry about things like which tool is used, format etc. Only look at the prompts/instructions given to the tools in each step and the step dependencies. * Assume that in each step, the instructions that are given to the tool in the prompt, run perfectly as intended and give the ideal output as per the prompt. * Under that assumption, judge whether the final output of the plan would answer the original query well or not. * Here are some possible scores to assign: * **0.0:** If the plan is completely irrelevant to the query. * **0.5:** If the plan partially answers the query but has some gaps/flaws. * **1.0:** If the plan perfectly and completely addresses all aspects of the query. Additional Information: - Placeholders: There are 2 types of placeholders: - Step placeholder (for e.g (1), (2)) which gets replaced by the output of the corresponding step during execution. - Query placeholder i.e. (query) which gets replaced by the original query during execution. - So wherever in the plan you see something like (1), (2), (query) etc, assume that is replaced by their corresponding above mentioned values. - The terms "call_ids" and "interaction_ids" are interchangeable. What matters is what comes after it, i.e. "call_ids of calls" or "interaction_ids of calls" would both fetch the ids of CALLS. Similarly, "call_ids of interactions" and "interaction_ids of interactions" would both fetch the ids of interactions. - In the final step, if the prompt itself does a good job at describing what is asked in the query, then it is fine even if it doesn't have the placeholder of the query. Input Format: You will be provided with a query and the corresponding plan. Output Format: You must provide a detailed explanation of your analysis related to that point, and then give the final score as per your verdict. Follow the below provided format: <explaination> | <score> | Here are some reference examples: {QUERY_ADHERENCE_EXAMPLES}

Table 31: System Prompts Used in **Metric-Wise Evaluator** (Step-Executability and Query-Adherence)

Module	Prompt
Tool-Usage Completeness (Metric-wise Evaluator)	**You are an expert plan evaluation agent** You will be given a plan along with the best possible plan and your task is to meticulously evaluate its tool-usage completeness and assign it a score. * Here is the format of a plan: Plan: { <step number>: {"step/query": <tool name>("<input instruction/prompt to the tool>"), "depends_on": [<step numbers that this step depends on>]} .... } Here are the tools at your disposal: {TOOL_DESCRIPTIONS} ## TOOL DESCRIPTION ENDS HERE Here are the things to keep in mind: * Look at the best possible plan and see if it has two steps with same prompt but different tool. For e.g in the following plan { 1: {"step": T2S([], "Fetch call_ids of calls related to payment adjustment inquiries."), "depends_on": []}, 2: {"step": RAG((1), "What patterns of customer sentiment emerge in these calls?"), "depends_on": [1]}, 3: {"step": T2S((1), "What patterns of customer sentiment emerge in these calls?"), "depends_on": [1]}, 4: {"step": LLM("Synthesize and summarize your answer to the query (query) using outputs from the following: (sub-query 2) using (tool 2): (2), (sub-query 3) using (tool 3): (3)"), "depends_on": [2, 3]} } * If this kind of a thing does not exist in the best possible plan: - directly return total steps - 0 = total steps. * If this kind of thing IS present in the best possible plan, then: - If the exact same thing (same 2 steps with same prompt and different tools) is present in the plan to be evaluated: - that is NOT a tool-usage completeness violation. - If neither of the two steps is present in the plan to be evaluated: - that is NOT tool-usage completeness violation. - If only 1 of the 2 steps are present in the plan to be evaluated: - that is a tool-usage completeness violation. Return the total steps - total number of tool-usage completeness violations. Input Format: You will be provided with 2 plans, the plan to be evaluated and the best possible plan. Output Format: You must provide a detailed explanation of your analysis related to that point, and then give the final score as per your verdict. Follow the below provided format: <explaination> | <score> | Here are some reference examples: {TOOL_USAGE_COMPLETENESS_EXAMPLES_WR}
Tool-Prompt Alignment (Metric-wise Evaluator)	**You are an expert plan evaluation agent** You will be given a plan along with the best possible plan and your task is to meticulously evaluate its tool-prompt alignment and assign it a score. * Here is the format of a plan: Plan: { <step number>: {"step/query": <tool name>("<input instruction/prompt to the tool>"), "depends_on": [<step numbers that this step depends on>]} .... } Here are the tools at your disposal: {TOOL_DESCRIPTIONS} ## TOOL DESCRIPTION ENDS HERE Here are the things to keep in mind: * For each step, look at the tool and the prompt. * If the instruction/prompt in a step cannot be fulfilled/answered by the assigned tool, then that step has a tool-prompt alignment violation. * If the prompt is complex and asks the tool to do multiple things at the same time, if even one of those things is outside the scope of the provided tool, then that step has a tool-prompt alignment violation. * For example if the prompt asks how sentiment shifted during the call, or any other change during the call, that cannot be answered using T2S and can only be answered using RAG. Return the total steps - total number of steps that have tool-prompt violation. Input Format: You will be provided with 2 plans, the plan to be evaluated and the best possible plan. Output Format: You must provide a detailed explanation of your analysis related to that point, and then give the final score as per your verdict. Follow the below provided format: <explanation> | <score> | Here are some reference examples: {TOOL_PROMPT_ALIGNMENT_EXAMPLES}

Table 32: System Prompts Used in **Metric-Wise Evaluator** (Tool-Prompt Alignment and Tool-Usage Completeness)

Prompt

You are a critical evaluator of structured, tool-augmented LLM plans.
Each plan is a sequence of steps designed to answer a user query using specialized tools. Your task is to compare a candidate plan (P) against the best possible reference plan (P*) and rate the candidate plan on a 7-point quality scale. You must reason carefully across **7 evaluation dimensions** and assign a rating based on how close P is to P* in terms of quality, structure, and correctness.
PLAN FORMAT Each plan is a dictionary where each key is a step number, and each value is an object with: - ‘"step"‘ or ‘"query"‘: containing a tool invocation of the form ‘TOOL_NAME(placeholder_inputs, prompt)‘ - ‘"depends_on"‘: a list of step numbers this step depends on #### Example Format: {{ 1: {{"step": T2S([], "Fetch interaction_ids of unresolved calls"), "depends_on": []}}, 2: {{"step": RAG((1), "Fetch calls where the sentiment transitioned from negative to positive within the transcript"), "depends_on": [1]}}, 3: {{"step": LLM("Fetch the interaction_ids as a list from the ‘data insights‘ table in (2)."), "depends_on": [2]}}, 4: {{"step": T2S((3), "Retrieve agent QA scores for resolution procedures in these calls."), "depends_on": [3]}} }}
PLACEHOLDER CONVENTIONS Placeholders are used in prompts to represent inputs to tools. These must follow standard conventions: - (query): The original user query - (<step number>): Refers to the output of a previous step - (tool <step number>): Refers to the tool used in that step (T2S, RAG, or LLM) - (sub-query <step number>): Refers to the prompt/query used in that step Placeholders must be well-formed and accurate to correctly wire dependencies between steps.
TOOL DESCRIPTIONS {TOOL_DESCRIPTIONS}
EVALUATION DIMENSIONS Evaluate the candidate plan (P) against the gold plan (P*) using these criteria: 1. Precision What percentage of steps in the plan (P) are present in the best possible plan (P*)? 2. Recall What percentage of steps in the best possible plan (P*) are present in the plan (P)? 3. F1 Score What percentage of steps in the best possible plan (P*) are present in the plan (P)? 4. Format Correctness Is the plan valid JSON with correct keys ("step" or "query" and "depends_on")? (a) Is the overall plan valid JSON? (b) What percentage of steps in the plan (P) have correct format - "step" or "query" and "depends_on" 5. Dependencies What percentage of steps in the plan (P) have correct dependencies? 6. Placeholders What percentage of steps in the plan (P) have correct placeholders?
INSTRUCTIONS - Be objective. Evaluate the candidate plan only in comparison to the gold plan. - Think through each metric and generate a detailed rationale. - Then assign a rating based on overall closeness on a 7-point scale: {{Extremely Bad, Very Bad, Bad, Acceptable, Good, Very Good, Extremely Good}}
Points to consider while evaluating the plan: - (a) If the plan (P) is not valid JSON, then it is automatically an "Extremely Bad" plan as it becomes difficult to evaluate the plan in terms of precision, recall, F1 score, format correctness, dependencies, and placeholders. - (b) If the plan (P) is valid JSON, then we can evaluate the plan in terms of precision, recall, F1 score, format correctness, dependencies, and placeholders. - (c) The individual metrics such as precision, recall, F1 score, format correctness, dependencies, and placeholders can be computed objectively. However, the final rating can be subjective but strictly based on the individual metrics which capture how close the candidate plan (P) is to the gold plan (P*). - (d) The metrics precision, recall and F1 score are computed by comparing the candidate plan (P) with the gold plan (P*). In contrast, the metrics format correctness, dependencies, and placeholders are computed by simply checking the plan (P) against the required format, dependencies and placeholders. - (e) While the final rating is subjective, you can follow the following guidelines provided the overall format of the plan is correct (valid json): - If f1_score is > 95%, then the rating is "Extremely Good" - If f1_score is > 85%, then the rating is "Very Good" - If f1_score is > 75%, then the rating is "Good" - If f1_score is > 60%, then the rating is "Acceptable" - If f1_score is > 45%, then the rating is "Bad" - If f1_score is > 30%, then the rating is "Very Bad" - If f1_score is < 30%, then the rating is "Extremely Bad"
Points to keep in mind while checking the match between P and P*: - (a) The steps in P and P* should have the exact same semantics and interpretation. If multiple steps in P* are carried out in a single step in P, then the step in P is incorrect. - (b) The steps in P and P* should have the same tool, format, placeholders and dependencies for them to be considered a match. Only if this condition is met, we can say that both the steps have the same semantics and interpretation. - (c) The steps in P and P* need not exactly match in terms of the prompt. The requirement is that the prompts in both the steps in P and P* should be semantically similar and have the same intent.
OUTPUT FORMAT {{ "rationale": {{ "precision": <rationale for precision>, "recall": <rationale for recall>, "f1_score": <rationale for f1_score>, "format_correctness": <rationale for format_correctness>, "dependencies": <rationale for dependencies>, "placeholders": <rationale for placeholders>, "rating": <rationale for rating> }} "score": {{ "precision": <score for precision>, "recall": <score for recall>, "f1_score": <score for f1_score>, "format_correctness": <score for format_correctness>, "dependencies": <score for dependencies>, "placeholders": <score for placeholders>, "rating": <score for rating> }} }} Note: The output should be a valid JSON object.

Table 33: System Prompt Used For **One-Shot Evaluator**

Prompt

You are an impartial Judge LLM. Evaluate three candidate responses (R1, R2, R3) to a given Query using the rubric below. Rely only on the provided text. Do not add external facts or assumptions.
Rubric (score each response independently)
(a) Validity (0/1): 0 = invalid (e.g., "cannot be answered due to no data," system error messages, or refusal). 1 = valid (any other kind of substantive response).
(b) Consistency (0/1): Is the response answering the given Query?
A response cannot be consistent if it is invalid.
Valid but off-topic $\implies$ consistency = 0.
Valid and on-topic $\implies$ consistency = 1.
(c) Completeness (1-7): How fully does the response address the Query with relevant details?
Consider only details relevant to the Query.
1 = Extremely Bad, 7 = Extremely Good.
(d) Redundancy (1-7): How free is the response from irrelevant or unrelated details?
7 = no fluff, to the point.
The more irrelevant content, the lower the score.
Gating rules
- Only valid responses can be consistent or complete.
- Only valid and consistent responses can be complete.
- For invalid responses, set: consistency = 0, completeness = 1, redundancy = 7.
Decision set (pick exactly one)
{"Neither", "R1", "R2", "R3", "R1,R2", "R1,R3", "R2,R3", "All"}
Use the following tie logic:
- Disqualify any response with validity = 0 or consistency = 0.
- If none remain, output "Neither".
- Among remaining, compare by (completeness, redundancy) in lexicographic order (higher is better).
- If all three tie on both metrics $\implies$ "All".
- If exactly two tie for best, then return that pair (e.g., "R1,R3").
- Otherwise return the single best (e.g., "R2").
Guidelines for scoring
- Do not focus too much on the format of the response being scored. As long as the response is valid and consistent, the format can be a simple list or dictionary and not necessary textual. - Example: - Query: "What percentage of calls required a supervisor escalation, and what is the average QA score for such calls over the last year?" - Response 1: {'pct_escalated_calls': Decimal('12.326849'), 'avg_escalated_qa_score': 0.9269446373201656} - This is a valid and consistent response. The format is a dictionary and not textual. Despite the format, the response is complete and accurate. - Response 2: "' # Analysis of Supervisor Escalations and QA Scores Based on the analysis, I can provide some interesting insights about supervisor escalations and their relationship with QA scores:
- Only 0.11% of all interactions required supervisor escalation over the past year - The average QA score for escalated calls was 94.00 - For comparison, non-escalated calls had an average QA score of 96.33
While escalated calls do show slightly lower QA scores, the difference is relatively small (only 2.33 points lower), suggesting that agents generally maintain good quality standards even in challenging situations that require escalation. "' - This is a valid and consistent response. However, there's a lot of fluff in the response. In addition, the response is redundant as it contains information about non-escalated calls which is not relevant to the Query.
- Do not focus on the correctness of the numeric values such as counts, percentages, average scores etc. because you are not provided with the ground truth. Simply because there's an alignment in the numeric values between two of the three responses does not mean that the third response is incorrect.
Output format
Return only a single JSON object (no prose, no markdown) with this exact schema and key order:
{ "rationale": "<2-4 sentence high-level justification that explains your reasoning for how the responses compare>", "R1": {"validity": <0|1>, "consistency": <0|1>, "completeness": <1-7>, "redundancy": <1-7>}, "R2": {"validity": <0|1>, "consistency": <0|1>, "completeness": <1-7>, "redundancy": <1-7>}, "R3": {"validity": <0|1>, "consistency": <0|1>, "completeness": <1-7>, "redundancy": <1-7>}, "overall_decision": "<one of the allowed strings>" }
Process requirements
- First, analyze the three responses and produce the rationale (2-4 sentences).
- Then assign the four metric scores (validity, consistency, completeness, redundancy) for R1, R2, R3.
- Finally, generate the overall_decision strictly from the allowed set.

Table 34: System prompt for the Judge LLM used to evaluate and compare R1/R2/R3 responses in the end-to-end QA study.

Stage	Generated Output	Annotated Output
Query Generated	What are customers most dissatisfied about in refund-related calls longer than 30 minutes?	N/A

Initial Plan Generated	{
	1: {"step": T2S([], "Fetch interaction_ids of interactions where the call driver is related to refunds and duration is longer than 30 minutes."), "depends_on": []},
	2: {"step": RAG((1), "What are customers most dissatisfied about?"), "depends_on": [1]}
	}
		N/A

Metric-Wise and One-Shot Evaluator Score for Initial Plan:	1. Format: 20 2. Tool-Prompt Alignment: 15 3. Step-Atomicity: 7.5 4. Query-Adherence: 12 5. Dependencies: 10 6. Redundancies: 10 7. Tool-Usage Completeness: 5 8. Total (Metric-Wise): 79.5 9. One-Shot: 81.5 (Good)	N/A
Step-Evaluator Output: On **1st** pass through feedback loop and step to be judged is **Step 1**	Step: 1 1. Nothing is wrong in this step and it can be executed normally : NO CHANGE	Step: 1 1. The query asks for "refund-related calls" specifically whereas the step filters for all interactions. So the "interaction_ids" needs to be changed to "call_ids" and "interactions" needs to be changed to "calls": INCORRECT PROMPT

Plan-Optimizer Output: On **1st** pass through feedback loop. Recieves the input from previous step-evaluator output	"CHANGE 0": Based on the output of the step-wise evaluator, the step requires no change.
	"CHANGE 1": Since there is no change in change 0, it is clear that there won't be any changes in the change 1 either and the plan remains unchanged.
	"NEW PLAN STARTS":
	{
	1: {"step": T2S([], "Fetch interaction_ids of interactions where the call driver is related to refunds and duration is longer than 30 minutes."), "depends_on": []},
	2: {"step": RAG((1), "What are customers most dissatisfied about?"), "depends_on": [1]}
	}
		"CHANGE 0": Based on the output of the step-wise evaluator, the terms "interaction_ids" and "interactions" need to be changed "call_ids" and "calls".
		"CHANGE 1": The rest of the plan makes perfect sense with this change alone, so no additional change 1 needs to be made.
		"NEW PLAN STARTS":
		{
		1: {"step": T2S([], "Fetch call_ids of calls where the call driver is related to refunds and duration is longer than 30 minutes."), "depends_on": []},
		2: {"step": RAG((1), "What are customers most dissatisfied about?"), "depends_on": [1]}
		}

Step-Evaluator Output: On **2nd** pass through feedback loop and step to be judged is **Step 2**	Step: 2. 1. Reasons for customer dissatifaction can be found out using T2S as well. So this step should be performed by T2S along with RAG : MULTI-TOOL PROMPT	Step: 2. 1. The step itself is fine, but reasons for customer dissatifaction can be found out using the call driver data in T2S as well. So this step should be performed by both the tools. : MULTI-TOOL PROMPT

Table 35: **End-to-End Lineage Guided Decomposition Evaluation Trace:** Stage-wise Generated Outputs vs. Human-Annotated Plans

Stage	Generated Output	Annotated Output

Plan-Optimizer Output: On **2nd** pass through feedback loop. Recieves the input from previous step-evaluator output	"CHANGE 0": As per the step-evaluator output, an additional step needs to be added with the same prompt as that of step 3 but with T2S as the tool.
	"CHANGE 1": Since we are adding this new step with the same prompt but different tool, we also need to add an LLM aggregation step to combine the outputs from these 2 steps into the final output that the user will receive.
	"NEW PLAN STARTS":
	{
	1: {"step": T2S([], "Fetch interaction_ids of interactions where the call driver is related to refunds and duration is longer than 30 minutes."), "depends_on": []},
	2: {"step": RAG((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
	3: {"step": T2S((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
	4: {"step": LLM("Combine the following 2 outputs to answer the following query (query):First:(2)Second:(3)"), "depends_on": [2, 3]},
	}
		"CHANGE 0": According to the step-evaluator's feedback, an extra step should be inserted using the same prompt as step 3, but with T2S selected as the tool.
		"CHANGE 1": Since this new step uses the same prompt but a different tool, an additional LLM aggregation step must be included to merge the outputs from both steps into the final response that will be delivered to the user.
		"NEW PLAN STARTS":
		{
		1: {"step": T2S([], "Fetch call_ids of calls where the call driver is related to refunds and duration is longer than 30 minutes."), "depends_on": []},
		2: {"step": RAG((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
		3: {"step": T2S((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
		4: {"step": LLM("Combine the following 2 outputs to answer the following query (query):First:(2)Second:(3)"), "depends_on": [2, 3]},
		}

Step-Evaluator Output: On **3rd** pass through feedback loop and step to be judged is **Step 3**	Step: 3 1. The step itself is fine, but reasons for customer dissatifaction can be found out using RAG as well. So this step should be performed by both the tools. : MULTI-TOOL PROMPT	Step: 3 1. The step itself is appropriate, but since reasons for customer dissatisfaction can also be identified using RAG, this step should be executed by both tools. : MULTI-TOOL PROMPT

Plan-Optimizer Output: On **3rd** pass through feedback loop. Receives the input from previous step-evaluator output	"CHANGE 0": Looking at the step-evaluator and the old plan, it seems that the step is already being performed by RAG in the previous step. So, nothing needs to be done to the plan and it can remain as it is.
	"CHANGE 1": Since there is no change 0, we don't need any change 1 either.
	"NEW PLAN STARTS":
	{
	1: {"step": T2S([], "Fetch interaction_ids of interactions where the call driver is related to refunds and duration is longer than 30 minutes."), "depends_on": []},
	2: {"step": RAG((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
	3: {"step": T2S((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
	4: {"step": LLM("Combine the following 2 outputs to answer the following query (query):First:(2)Second:(3)"), "depends_on": [2, 3]},
	}
		"CHANGE 0": Based on the step-evaluator output and the existing plan, this step is already being handled by RAG in the preceding step. Therefore, no modifications are required, and the plan can remain unchanged.
		"CHANGE 1": As no change 0 is needed, there is likewise no need for any change 1.
		"NEW PLAN STARTS":
		{
		1: {"step": T2S([], "Fetch call_ids of calls where the call driver is related to refunds and duration is longer than 30 minutes."), "depends_on": []},
		2: {"step": RAG((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
		3: {"step": T2S((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
		4: {"step": LLM("Combine the following 2 outputs to answer the following query (query):First:(2)Second:(3)"), "depends_on": [2, 3]},
		}

Table 36: **End-to-End Lineage Guided Decomposition Evaluation Trace:** Stage-wise Generated Outputs vs. Human-Annotated Plans (Contd.)

Stage	Generated Output	Annotated Output
Step-Evaluator Output: On **4th** pass through feedback loop and step to be judged is **Step 4**	Step: 4 1. The step is appropriate and requires no additional changes. : NO CHANGE	Step: 4 1. The step is perfectly fine and executable as it is and does not require any further modifications. : NO CHANGE

Plan-Optimizer Output: On **4th** pass through feedback loop. Recieves the input from previous step-evaluator output	"CHANGE 0": As per the step-evaluator output, no adjustment is required.
	"CHANGE 1": Since there is no change 0, we don't need any change 1 either.
	"NEW PLAN STARTS":
	{
	1: {"step": T2S([], "Fetch interaction_ids of interactions where the call driver is related to refunds and duration is longer than 30 minutes."), "depends_on": []},
	2: {"step": RAG((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
	3: {"step": T2S((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
	4: {"step": LLM("Combine the following 2 outputs to answer the following query (query):First:(2)Second:(3)"), "depends_on": [2, 3]},
	}
		"CHANGE 0": According to the step-evaluator output, no modifications are necessary.
		"CHANGE 1": As no change 0 is required, there is likewise no need for a change 1.
		"NEW PLAN STARTS":
		{
		1: {"step": T2S([], "Fetch call_ids of calls where the call driver is related to refunds and duration is longer than 30 minutes."), "depends_on": []},
		2: {"step": RAG((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
		3: {"step": T2S((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
		4: {"step": LLM("Combine the following 2 outputs to answer the following query (query):First:(2)Second:(3)"), "depends_on": [2, 3]},
		}

Final Plan: Gotten upon doing Human Annotation on the feed-back loop output plan. In this case, "interaction_ids" was changed to "call_ids" in step 1 since the query specifically asks for calls. This is a mistake that the feedback-loop failed to fix.	{
	1: {"step": T2S([], "Fetch call_ids of calls where the call driver is related to refunds and duration is longer than 30 minutes."), "depends_on": []},
	2: {"step": RAG((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
	3: {"step": T2S((1), "What are customers most dissatisfied about?"), "depends_on": [1]},
	4: {"step": LLM("Combine the following 2 outputs to answer the following query (query):First:(2)Second:(3)"), "depends_on": [2, 3]},
	}
		N/A

Metric-Wise and One-Shot Evaluator Score for Final Plan	1. Format: 20 2. Tool-Prompt Alignment: 20 3. Step-Atomicity: 15 4. Query-Adherence: 15 5. Dependencies: 10 6. Redundancies: 10 7. Tool-Usage Completeness: 10 8. Total (Metric-Wise): 100 9. One-Shot: 98 (Extremely Good)	1. Format: 20 2. Tool-Prompt Alignment: 20 3. Step-Atomicity: 15 4. Query-Adherence: 15 5. Dependencies: 10 6. Redundancies: 10 7. Tool-Usage Completeness: 10 8. Total (Metric-Wise): 100 9. One-Shot: 100 (Extremely Good)

Table 37: **End-to-End Lineage Guided Decomposition Evaluation Trace:** Stage-wise Generated Outputs vs. Human-Annotated Plans (Contd.)

Query Type	Query
Simple	What are the main reasons behind interactions that are more than 1 hour long?
Compound	Show interaction count breakout by entity type for last 60 days and for each entity type give details of agent who attended most number of interactions also for those agents give details of their teams
Subjective	What reasons are customers providing for dissatisfaction in calls longer than 45 minutes?
Objective	How do customer concerns and issues change during 18-24 Nov compared to 25-30 Nov?
Zero-Hop	What is the average call duration in the last week?
One-Hop	What kind of feedback are customers giving about agent empathy during their calls?
Two-Hop	Which issues trigger longer calls related to payment processing concerns, and how are agents alleviating customer frustrations during these calls?
Three-Plus Hop	Identify patterns in customer dissatisfaction by extracting the most common phrases expressed in negative sentiment calls related to refunds and financial adjustments, and correlate these with the average QA scores of associated agents.

Table 38: Query Examples Broken Down By **Subjective/Objective, Simple/Compound and Number of Hops.**

Algorithm 1 Planner Feedback Loop: Step-wise Evaluator → Plan Optimizer

Require: Initial plan $P^{(0)} = \{p_1, \ldots, p_n\}$
Ensure: Optimized plan $P^\star$ and full lineage $\mathcal{L}$

$P \leftarrow P^{(0)}$; $\mathcal{L} \leftarrow [P]$ ▷ store lineage states
$max\_passes \leftarrow M$ ▷ e.g., M=4
$pass \leftarrow 0$; $changed \leftarrow$ **true**
while $changed$ = **true and** $pass < max\_passes$ **do**
  $changed \leftarrow$ **false**; $i \leftarrow 1$; $\ell \leftarrow$ length(P)
  while $i \leq \ell$ **do**
    $step \leftarrow P[i]$; $deps \leftarrow$ DEPSOF(P, i)
    $(tags, rationale) \leftarrow$ STEPWISEEVALUATE$(step, deps)$
    if $tags = \{$NOCHANGE$\}$ **then**
      $i \leftarrow i + 1$
    else
      $P' \leftarrow$ PLANOPTIMIZE$(P, i, tags, rationale)$
      if $P \neq P'$ **then**
        if length$(P') = \ell$ **then**
          $P \leftarrow P'$
          append P to $\mathcal{L}$
          $i \leftarrow i + 1$
        else
          $P \leftarrow P'$
          append P to $\mathcal{L}$
          $\ell \leftarrow$ length(P) ▷ re-check the same i on next loop
        end if
        $changed \leftarrow$ **true**
      else
        $i \leftarrow i + 1$
      end if
    end if
  end while
  $pass \leftarrow pass + 1$
end while
return $P^\star \leftarrow P, \mathcal{L}$

Lemma 1 (Termination)**.** *The feedback loop in Algorithm 1 terminates in at most M passes. Within each pass, the inner scan either (i) advances the step index i or (ii) applies a finite structural change to the plan that is immediately recorded and re-checked; the outer guard $pass < M$ guarantees termination.*

Proof sketch. Each pass is bounded by the guard

$pass < M$. Inside a pass, whenever the plan changes, the lineage is updated and the scan either advances to $i{+}1$ (no length change) or re-checks the same i (length change); when no changes occur for a full pass, the loop exits. Hence the procedure stops after at most M passes. □

C.5 Human Verification

Expert annotators review the final plan in each lineage; minor edits (if needed) produce the *best* plan. All lineage states are retained for analysis/training.

C.6 Number of Hops: Definition and Computation

We define *number of hops* from the dependency structure of the best possible plan for a query. Let the plan be a Directed Acyclic Graph (DAG) where each node is a step and edges point from a prerequisite step to its consumer.

Categories.

- **Zero-hop:** No dependencies. Direct tool calls (e.g., a single `T2S` or `RAG` step) produce the final answer; multiple independent producers also qualify if no step depends on another.
- **One-hop:** Exactly one dependency layer. Typical pattern: an `LLM` synthesis step depends on one or more independent producer steps (e.g., `T2S` and `RAG`), neither of which depends on prior steps.
- **Two-hop:** Two sequential dependency layers. Example: a final `LLM` step depends on `RAG`/`T2S` analysis steps, each of which depends on a `T2S` filtering step that scopes the interaction IDs.
- **Three-plus:** Three or more sequential dependency layers; e.g., filter → enrich → analyze → synthesize.

Operational computation. We compute hop count as the maximum dependency depth among steps that contribute to the final answer:

$$\text{depth}(s) = \begin{cases} 0, \text{ if depends\_on}(s) = \varnothing, \\ 1 + \max\limits_{p \in \text{depends\_on}(s)} \text{depth}(p) \end{cases} \tag{1}$$

$$\text{hops} = \max_{s \in S_{\text{final}}} \text{depth}(s). \tag{2}$$

where S_{final} are sink steps (no consumers) or designated answer-producing steps (e.g., the final `LLM` synthesis).

Examples.

1. `T2S([], "Compute average call duration last month")` ⇒ **zero-hop**.
2. `RAG((1), ...)` and `T2S((1), ...)` not used; instead two independent producers `RAG([], ...)`, `T2S([], ...)` feeding an `LLM` ⇒ **one-hop**.
3. `T2S([], "Fetch interaction_ids ...")` → `{RAG((1), ...), T2S((1), ...)}` → `LLM(...)` ⇒ **two-hop**.
4. Extend with another dependent analysis layer before synthesis ⇒ **three-plus**.

We compute this hop count algorithmically for each gold plan and use it as metadata when analyzing model performance by reasoning depth.

C.7 Module Set and Validation Protocols

We employ four LLM-driven modules:

1. **Metric-wise evaluator** (7 metrics; see §4).
2. **One-shot evaluator** (reference-based: precision/recall/F1 + format; 7-point rating; see §4)
3. **Step-wise evaluator** (diagnostics tags; see §3).
4. **Plan optimizer** (Change 0/1; lineage revision; see §3).

Their prompts and evaluation against human ground truth are reported in Tables 28–33 and Tables 21–26 respectively.

D Evaluation Details

D.1 Metric Definitions and Rubrics

All seven metrics are computed by LLM evaluators using task-specific rubric prompts. Below we summarize decision rules; full prompts are included in Tables 30–32.

M1: Tool–Prompt Alignment (20 pts). Checks that the step's prompt lies within the assigned tool's capabilities. Penalize: (i) sentiment *changes within calls* assigned to T2S (use RAG instead); (ii) filtering calls based on their duration assigned to RAG (use T2S instead); (iii) prompting LLM to answer questions related to QA scores of agents (use T2S instead). Scoring: The LLM evaluator assesses each step as pass or fail, computes the total number of passed steps, divides by the total number of steps, and then scales the result to a maximum of 20.

M2: Format Correctness (20 pts). Plan must be JSON parseable. Every dependent step must contain a correct numeric placeholder (e.g., `(2)`); quotes and parentheses must balance. Four standard placeholders are defined which are as follows:

- `(query)`: Refers to the original query.
- `(<step_id>)`: Refers to the output from step <step_id>.
- `(tool <step_id>))`: Refers to the name of the tool {T2S, RAG, LLM} assigned to the step <step_id>).
- `(sub-query <step_id>)`: Refers to the prompt passed to the tool assigned to the step <step_id>.

The aforementioned standard placeholders should be used correctly in the plan. Scoring: The LLM evaluator assesses each step as pass or fail based on the above rules, computes the total number of passed steps, divides by the total number of steps, and then scales the result to a maximum of 20.

M3: Step Executability / Atomicity (15 pts). Each step should be executable using a single tool call. Compound prompts—such as those given to RAG (e.g., "How did sentiment change *and* what did agents do?") or to T2S (e.g., "Analyze the trends in customer sentiment scores based on the primary call driver categories over the past six months.")—are penalized. Compound prompts sent to RAG often produce incomplete responses, while those sent to T2S can result in timeouts, as complex instructions translate into SQL queries that are prone to exceeding execution limits in Snowflake. Steps that are explicitly decomposed into atomic units are rewarded.

Scoring: The LLM evaluator marks each step as pass or fail according to these rules, calculates the total number of passed steps, divides by the total number of steps, and scales the result to a maximum of 15 points.

M4: Query Adherence (15 pts). Judge whether the plan, if executed perfectly, answers the query fully and uses the correct modality ("calls" vs. "interactions").

Scoring: The LLM evaluator returns a score in $\{0, 0.5, 1\}$ depending on how well the generated plan adheres to the given query which is then scaled to a maximum of 15 points.

M5: Dependencies (10 pts). Every step that uses prior outputs has `depends_on` populated and the prompt includes the correct placeholder(s). Penalize missing/incorrect placeholders and unnecessary edges.

Scoring: The LLM evaluator marks each step as pass or fail according to these rules, calculates the total number of passed steps, divides by the total number of steps, and scales the result to a maximum of 10 points.

M6: Redundancy (10 pts). Detect duplicated work or repeated filters. Typical anti-pattern: T2S step (1) filters "delivery delay," followed by T2S(1) that again says "for the calls with delivery delay," causing unnecessary joins. Deduct per occurrence.

Scoring: The LLM evaluator marks each step as pass or fail according to these rules, calculates the total number of passed steps, divides by the total number of steps, and scales the result to a maximum of 10 points.

M7: Tool-Usage Completeness (10 pts). When a step's goal clearly merits dual-tool coverage (e.g., sentiments exist in transcripts and structured summaries), penalize plans using only one.

Scoring: The LLM evaluator marks each step as pass or fail depending on whether there's a violation with respect to tool-usage completeness, calculates the total number of passed steps, divides by the total number of steps, and scales the result to a maximum of 10 points.

Reference-free vs. with-reference split. All metrics are reference-based, as leveraging the reference (best possible) plan significantly enhances their performance. Table 21 provides a comparison between reference-based and reference-free evaluators.

D.2 One-Shot Overall Evaluation via Judge LLM

We evaluate a candidate plan P against the best plan $P^\star$ using a Judge LLM prompted to compute:

- **Precision:** % of steps in P present in $P^\star$
- **Recall:** % of steps in $P^\star$ present in P
- $\mathbf{F}_1$

The following are assessed on P alone: **Format Correctness**, **Dependencies** & **Placeholders**. The Judge LLM then assigns a seven-point rating based on these scores, with the rule that non-JSON plans are rated *Extremely Bad* irrespective of F_1.

Prompts. We use the following prompts (redacted for brevity here; full text in Table 33):

- **System:** defines plan format, placeholder conventions, tool descriptions, six dimensions to score, rating rubric, and output schema.
- **User:** provides the query, candidate P, and best plan $P^\star$, plus reference examples.

The Judge LLM computes step matches and metrics per rubric in the system prompt (no hand-coded matching in our pipeline). Output must be valid JSON with both a rationale block and a score block.

D.3 Aggregation and Adjudication

All evaluators are LLM-based with deterministic decoding (Refer Table 50 for LLM configurations). A random sample per split is manually audited to calibrate thresholds and check stability across model updates.

D.4 Learning Metric Weights

Setting. Each plan P is scored on $M{=}7$ metrics, where the LLM evaluators return raw sub-scores $s_m(P) \in [0,1]$ for $m \in \{1,\dots,7\}$. We learn nonnegative *weights* $w_m \geq 0$ that represent the maximum points allocated to each metric. The total score is

$$S(P;\mathbf{w}) \;=\; \sum_{m=1}^{7} w_m\, s_m(P).$$

We partition metrics into two groups: Effectiveness $\mathcal{E}$ (4 metrics) and Efficiency $\mathcal{F}$ (3 metrics).

Constraints (fixed group budgets). We enforce point budgets at the group level,

$$\sum_{m\in\mathcal{E}} w_m \;=\; B_{\mathcal{E}}, \qquad \sum_{m\in\mathcal{F}} w_m \;=\; B_{\mathcal{F}},$$

with $B_{\mathcal{E}}{=}70$ and $B_{\mathcal{F}}{=}30$ in our experiments.[4]

Monotonic lineage objective. For each validation query q, we consider a lineage triple $\big(P^{(q)}_{\text{best}},\, P^{(q)}_{\text{pen}},\, P^{(q)}_{\text{ante}}\big)$ derived from human annotations, where the intended aggregate ordering is best $\succ$ pen $\succ$ ante. This ordering *need not* hold metric-wise; it is enforced only at the weighted sum level. We seek weights $\mathbf{w}$ that maximize a global margin $\gamma \geq 0$ while imposing strict, margin-based monotonicity:

$$S\big(P^{(q)}_{\text{best}};\mathbf{w}\big) \geq S\big(P^{(q)}_{\text{pen}};\mathbf{w}\big) + \gamma, \quad \forall q,$$
$$S\big(P^{(q)}_{\text{pen}};\mathbf{w}\big) \geq S\big(P^{(q)}_{\text{ante}};\mathbf{w}\big) + \gamma, \quad \forall q.$$

Let $\Delta s^{(q)}_{A\triangleright B,m} = s_m(P^{(q)}_A) - s_m(P^{(q)}_B)$.

Optimization program. We solve the following linear program (LP):

$$\max_{w,\gamma} \quad \gamma \tag{3}$$
$$\text{s.t.} \quad \sum_{m\in\mathcal{E}} w_m = B_{\mathcal{E}}, \tag{4}$$
$$\sum_{m\in\mathcal{F}} w_m = B_{\mathcal{F}}, \tag{5}$$
$$w_m \geq 0 \qquad \forall m, \tag{6}$$
$$\sum_{m=1}^{7} w_m\, \Delta s^{(q)}_{\text{best,pen},m} \geq \gamma \qquad \forall q, \tag{7}$$
$$\sum_{m=1}^{7} w_m\, \Delta s^{(q)}_{\text{pen,ante},m} \geq \gamma \qquad \forall q. \tag{8}$$

If the LP is infeasible (rare; e.g., due to noisy labels), we solve a hinge-relaxed variant with slacks $\xi_q, \zeta_q \geq 0$ and penalty $C > 0$:

[4] Equivalently, one can work on the probability simplex with a 70:30 split and rescale to points post hoc.

$$\max_{w,\gamma,\xi,\zeta} \quad \gamma - C\sum_{q}(\xi_q + \zeta_q) \tag{9}$$

$$\text{s.t.} \quad \sum_{m\in\mathcal{E}} w_m = B_{\mathcal{E}}, \tag{10}$$

$$\sum_{m\in\mathcal{F}} w_m = B_{\mathcal{F}}, \tag{11}$$

$$w_m \geq 0 \qquad \forall m, \tag{12}$$

$$\sum_{m=1}^{7} w_m \, \Delta s^{(q)}_{\text{best,pen},m} \geq \gamma - \xi_q, \qquad \forall q, \tag{13}$$

$$\xi_q \geq 0 \qquad \forall q, \tag{14}$$

$$\sum_{m=1}^{7} w_m \, \Delta s^{(q)}_{\text{pen,ante},m} \geq \gamma - \zeta_q, \qquad \forall q, \tag{15}$$

$$\zeta_q \geq 0 \qquad \forall q. \tag{16}$$

Practical procedure and quantization. We implement a coarse grid search on each group simplex with step size 0.02, selecting $\mathbf{w}$ that maximizes the number of satisfied inequalities and, secondarily, the median margin. We then (i) rescale the group-wise weights to the point budgets $(B_{\mathcal{E}}, B_{\mathcal{F}})$ and (ii) quantize to the desired lattice (e.g., integers or multiples of 5) by greedy round-and-adjust while preserving group sums and non-negativity. We retain the quantized $\mathbf{w}$ with minimal deviation from the continuous solution and revalidate the margin constraints.

Final weights used. In all experiments we use the following quantized weights (total 100 points): *TP Alignment* $= 20$, *Format* $= 20$, *Step Executability* $= 15$, *Query Adherence* $= 15$, *Dependency* $= 10$, *Redundancy* $= 10$, *Tool-Usage Completeness* $= 10$. This matches the 70:30 Effectiveness/-Efficiency budget and is consistent with the learned proportions after quantization.

Why lineage-based weights? This objective encodes the intended ordinal relationship among the *top* plans per query, ensuring the evaluator's total score reflects true quality progression while respecting the Effectiveness/Efficiency budget. It reduces sensitivity to absolute metric scales and emphasizes rank-consistent scoring.

D.5 Weight Sensitivity Analysis For Metric-Wise Evaluator

Our main aggregate score S_{learned} uses metric-wise evaluator outputs and weights learned from human-preferred lineage triples under the 70:30 Effectiveness–Efficiency budget (App. D.4). To assess the robustness of planner rankings to this choice of weights, we conduct a small-scale sensitivity analysis on the same set of query–plan pairs used for metric-wise evaluation (14 LLMs $\times$ 2 prompt settings).

Schemes. For each plan, we first normalize each metric by its maximum possible value (Dependency, Format, Tool-Prompt Alignment, Redundancy, Tool-Usage Completeness, Step Executability, Query Adherence). We then consider:

1. **Equal-weights scheme**: each normalized metric receives the same weight, and the aggregate is the average of normalized scores rescaled to $[0, 100]$.

2. **Random-weight schemes**: we sample ten random weight vectors that preserve the 70:30 Effectiveness–Efficiency budget by drawing Dirichlet distributions separately over the Effectiveness metrics (TP Alignment, Format, Step Executability, Query Adherence) and the Efficiency metrics (Dependency, Redundancy, Tool-Usage Completeness). Each random draw yields a new aggregate score S_{rand}.

For each prompt type (with-lineage vs. without-lineage), we average scores per LLM and compute Spearman rank correlation between the rankings induced by S_{learned} (the score used in the main text) and those from the alternative schemes.

Results. Table 39 reports rank correlations. Under equal weights, planner rankings are very close to the learned-weight rankings ($\rho = 0.934$ with-lineage; $\rho = 0.894$ without-lineage). Across ten random draws that respect the 70:30 budget, the median correlation remains high ($\rho_{\text{med}} = 0.890$ with-lineage; $\rho_{\text{med}} = 0.842$ without-lineage), and even the worst random draws retain moderate agreement ($\rho_{\min} = 0.736$ and 0.503 respectively). Table 40 lists per-LLM scores under the learned, equal, and random-weight schemes. Overall, high-ranked LLMs remain high-ranked, and only minor reorderings occur among mid-tier models, indicating that our LLM-level conclusions are robust to

reasonable variations in metric weights and do not depend on a single, hand-picked setting.

D.6 End-to-End Correlation Study: North-Star vs. No-Plan Baseline

Setup. To test whether our planner-level metrics are predictive of end-to-end QA quality, we compare three systems:

- **R1 (No-plan baseline).** The existing production stack, which answers queries using our deployed T2S/RAG/LLM pipeline *without* an explicit planner. It directly returns a single final answer per query, and can be viewed as analogous to a plan-free ("no-lineage") baseline in this work.
- **R2 (North-star w/ human plans).** A north-star system that executes *human-annotated reference plans* (the same reference plans used in our benchmark) through the same T2S/RAG/LLM stack and merges structured and unstructured outputs.
- **R3 (North-star w/ LLM plans).** The same north-star stack driven by *LLM-generated plans* instead of human-annotated ones.

We evaluate on a stratified subset of 200 queries from the LLM-generated test set (the same pool used for plan evaluation) and, for completeness, on 100 real production queries; the latter only compare R1 and R3, since human-annotated plans do not exist for those queries. All three systems share the same underlying T2S and RAG tools and execution environment; **implementation details of these proprietary components are not released.**

Judge LLM and rubric. For each query, the three candidate responses (R1, R2, R3) are scored by an in-house Judge LLM that has been tuned for this task. The judge applies a fixed rubric with four metrics per response: *Validity* (0/1), *Consistency* (0/1; answers the query), *Completeness* (1–7), and *Redundancy* (1–7; higher is less fluff), together with gating rules (e.g., only valid and consistent responses can be complete). The system prompt used for the in-house Judge LLM for this task can be referred in Table 34. A deterministic decision rule then selects the best system(s) per query from the decision set {`R1`, `R2`, `R3`, `R1,R2`, ..., `Neither`}, based on lexicographic comparison of (Completeness, Redundancy) among valid, consistent responses.[5] Human annotators reviewed and, when necessary, edited approximately 30% of the judged cases to verify the quality of the automatic decisions. On a held-out set of 80 queries, two annotators independently selected the best system(s) among R1, R2, and R3; their raw agreement was 82.5% with Cohen's $\kappa = 0.69$, indicating high but not perfect human–human consistency. After adjudication, the Judge LLM's decisions matched the final human labels on 80.0% of queries with $\kappa = 0.65$, suggesting that the automatic comparison is well aligned with human preferences for this task while still leaving room for occasional disagreement.

Win-rate metric. For each system $S \in \{\text{R1}, \text{R2}, \text{R3}\}$ and query set, the *win rate* is defined as the fraction of queries for which S is included in the judge's *overall_decision* (e.g., `"R1,R3"` contributes a win to both R1 and R3). Because ties are allowed, win rates do not sum to 100%.

Results. Table 41 reports win rates on the LLM-generated test subset and on production queries. On the LLM-generated subset, the north-star system with human-annotated plans (R2) achieves the highest win rate (58.7%), substantially outperforming both the no-plan baseline (R1) and the north-star system with LLM-generated plans (R3). On production queries, the north-star system with LLM-generated plans (R3) outperforms the no-plan baseline (R1), despite using the same underlying tools.

Overall, these results provide end-to-end evidence that higher-quality plans (R2) lead to better final answers, and that even LLM-generated plans (R3) can outperform the no-plan baseline on real queries, supporting the practical relevance of our planner-level evaluation.

E Experimental Setup Details

E.1 Data Splits and Curation Protocol (Step-by-Step)

We summarize the sequential workflow used to construct plan lineages and gold plans, and to prepare train/validation/test splits.

Step 1: Stratified sampling from 600 queries. We first generated 600 domain queries (Sec. 3).

[5] The underlying Judge LLM and its training data are proprietary; we therefore release only the rubric and decision rule, not model details.

Prompt	ρ_{equal}	$\rho^{\text{min}}_{\text{rand}}$	$\rho^{\text{med}}_{\text{rand}}$	$\rho^{\text{max}}_{\text{rand}}$
With Lineage	0.934	0.736	0.890	0.947
Without Lineage	0.894	0.503	0.842	0.873

Table 39: Spearman rank correlations between LLM rankings under the learned aggregate score and alternative metric-weighting schemes: equal weights over normalized metrics, and ten random 70:30 Effectiveness–Efficiency draws.

LLM	Prompt	Learned	Equal	Rand (min)	Rand (median)	Rand (max)
claude-3-7-sonnet	With Lineage	84.80	77.01	70.70	79.36	87.36
llama4-maverick-17b-instruct	With Lineage	82.26	75.55	67.99	77.07	85.77
gpt-4o	With Lineage	81.47	74.37	67.56	75.43	83.78
claude-sonnet-4	With Lineage	79.90	72.64	65.34	73.26	80.82
llama3-3-70b-instruct	With Lineage	79.77	75.07	68.06	77.28	85.95
nova-pro	With Lineage	79.24	75.24	67.92	77.06	85.09
nova-micro	With Lineage	77.28	74.08	65.71	75.87	86.84
gpt-4o-mini	With Lineage	77.26	72.36	65.80	73.87	80.30
gpt-4.1-nano	With Lineage	77.16	71.13	64.63	72.35	78.33
o3-mini	With Lineage	71.85	68.49	57.02	71.13	77.48
claude-3-5-haiku	With Lineage	71.27	65.56	59.04	66.80	71.92
nova-lite	With Lineage	70.53	68.31	59.85	69.89	77.31
llama3-2-3b-instruct	With Lineage	70.51	66.51	57.26	66.64	77.71
llama3-2-1b-instruct	With Lineage	20.27	20.25	7.06	16.22	32.81
claude-3-7-sonnet	Without Lineage	83.33	75.90	71.17	81.45	85.74
gpt-4o	Without Lineage	82.82	74.90	69.81	79.74	83.43
gpt-4o-mini	Without Lineage	82.78	75.00	72.01	79.55	86.51
nova-pro	Without Lineage	82.70	75.82	70.55	80.20	88.05
llama4-maverick-17b-instruct	Without Lineage	82.50	74.88	72.07	80.03	83.49
nova-micro	Without Lineage	82.07	74.68	73.36	79.95	84.67
claude-3-5-haiku	Without Lineage	82.06	75.71	67.93	80.34	87.88
llama3-3-70b-instruct	Without Lineage	81.11	74.48	68.04	78.82	85.29
claude-sonnet-4	Without Lineage	77.98	71.67	58.04	75.33	82.27
gpt-4.1-nano	Without Lineage	76.05	69.14	61.63	72.35	80.47
llama3-2-3b-instruct	Without Lineage	75.84	68.06	67.31	72.91	76.54
nova-lite	Without Lineage	75.14	71.07	65.26	75.43	83.33
o3-mini	Without Lineage	74.06	69.75	57.67	73.00	82.04
llama3-2-1b-instruct	Without Lineage	56.79	50.39	44.88	51.87	57.40

Table 40: LLM scores under different metric-weighting schemes: baseline learned aggregate ("Learned") used in the main text, equal weights over normalized metrics ("Equal"), and ten random 70:30 Effectiveness–Efficiency draws ("Rand") summarized by their minimum, median, and maximum per planner. Rankings under alternative schemes remain highly correlated with the learned-weight ranking (Table 39).

From these, we sampled **100** queries using stratification across the key attributes introduced in the main text: (*i*) subjectivity (objective/subjective/very subjective), and (*ii*) structural complexity (simple/compound).

Step 2: Human-grounded lineage construction. For each of the 100 sampled queries:

1. **Initial plan.** We obtained a one-shot initial plan using *Nova-Lite*.

2. **Lineage simulation.** We *simulated* the outputs of the Step-wise Evaluator and Plan Optimizer via human annotation to produce a sequence of improving plans (the *plan lineage*) per query. This yields interpretable intermediate revisions mirroring the Evaluator→Optimizer loop (Appx. C.4).

3. **Gold verification.** The final plan in each lineage was verified by human annotators and, if needed, revised to obtain the **best possible (gold) plan**.

4. **Metric supervision for evaluators.** For each query, we took the first three plans *from the bottom* of the lineage (i.e., the three highest-quality plans) and annotated their *per-metric ranks* across the seven evaluation dimensions (Sec. 4). These annotations are used to *train/-tune* the metric-wise evaluators.

5. **One-shot supervision.** For each query, we

Data source	R1 total	Only R1	R1,R2	R1,R3	R2 total	Only R2	R2,R3	R3 total	Only R3	Neither	All
LLM-generated (N=200)	**42.75%**	24.64%	10.87%	7.25%	**58.70%**	31.16%	16.67%	**33.33%**	9.42%	0.00%	0.00%
Prod (N=100)	**50.94%**	28.30%	-	22.64%	-	-	-	**66.04%**	43.40%	5.66%	0.00%

Table 41: Breakdown of Judge-LLM decisions for the no-plan baseline (R1) and north-star systems (R2, R3). "R1 total" is the fraction of queries where R1 appears in the overall decision (i.e., in `R1`, `R1,R2`, or `R1,R3`); analogous definitions apply to "R2 total" and "R3 total". Columns such as "Only R1", "R1,R2", and "R1,R3" show the distribution over individual decision outcomes. Win rates do not sum to 100% because ties are allowed. R2 results are unavailable for live production queries due to the absence of human-annotated reference plans. To facilitate comparison, the systems in each row are color-coded by performance: blue denotes the highest win rate, magenta signifies the runner-up, and red indicates the lowest.

compared the *last but one* plan in the lineage against its gold plan (*last* plan in the lineage) and assigned a **7-point quality label** (Extremely Bad $\rightarrow$ Extremely Good) based on Precision/Recall/F_1 + format checks (Appx. D.2). These labels are used to *train/-tune* the one-shot overall evaluator.

Step 3: Train/validation partition for module tuning. From the 100 annotated queries, we sampled **20** for **Train** (prompt construction for all modules: metric-wise evaluators, one-shot Judge, step-wise evaluator, plan optimizer) and reserved the remaining **80** for **Validation**.

Step 4: Module validation against human ground truth. On the 80-query validation set, we evaluated each LLM-based module *against human annotations*:

1. **Metric-wise evaluator** (all seven metrics; Sec. 4).
2. **One-shot overall evaluator** (Precision/Recall/F_1 + 7-point rating).
3. **Step-wise evaluator** (diagnostic tags).
4. **Plan optimizer** (quality gains, format and dependency integrity).

Step 5: Test-time benchmarking and lineage usage. The **Test** split contains **500** queries. We use the finalized, optimized modules as follows:

- **Plan generation benchmarking.** For each query, we generate one-shot plans from **14 LLMs** under two settings: *without lineage* and *with lineage*. In the latter setting, the per-query lineage exemplars (derived via our feedback loop) are included in the prompt's reference examples.
- **Lineage for Nova-Lite only.** We run the iterative feedback loop to produce *new* lineages at test time *only* for Nova-Lite, to quantify end-to-end improvement relative to its one-shot baseline. For the other 13 models, we do *not* regenerate lineages at test time; they are evaluated with and without lineage exemplars in the prompt, but lineage construction is not part of their execution budget.

Proprietary data note. As detailed in Sec. 5, the full 600-query benchmark originates from Observe.AI's production environment and cannot be released. However, we publish a stratified 200-query subset (100 Train, 100 Test) with queries, reference plans, lineages, and per-planner evaluator scores; see Appx. E.2 for details. We also provide prompts and scoring rubrics to reproduce the pipeline on non-proprietary data.

E.2 Public Dataset Release

For reproducibility and qualitative analysis, we release a *separate* public dataset of 200 queries.[6] The dataset used for all reported results in the main paper is based on queries issued by real Observe.AI customers and cannot be shared for contractual and privacy reasons. Both the internal 600-query benchmark and the public 200-query release use **GPT-4o** to synthesize queries (Appx. E.5); the public subset is generated against a sandboxed synthetic test account so that no customer-specific business logic or identifiers appear, while preserving the same schema, tool palette, and evaluation setup.

The public dataset is split into:

- **Train (100 queries):** used analogously to the internal train/validation queries for prompt construction and calibration. Best plans and full plan lineages are human-annotated.
- **Test (100 queries):** held-out queries with human-annotated best plans. Plan lineages are obtained by running the iterative

[6]The public release is available at `https://github.com/Observeai-Research/tool-aware-planning-dataset/`.

	Subjectivity		Compoundness		# steps in BPP (grouped)		
Dataset	Subjective	Objective	Simple	Compound	[1,2]	[3,4]	[5,15]
Public (200)	105 (52.5%)	95 (47.5%)	106 (53.0%)	94 (47.0%)	68 (34.0%)	74 (37.0%)	58 (29.0%)
Main paper (600)	311 (51.8%)	289 (48.2%)	323 (53.8%)	277 (46.2%)	207 (34.5%)	223 (37.2%)	170 (28.3%)

Table 42: Comparison of marginal distributions for subjectivity, compoundness, and grouped best-plan length between the 200-query public release and the 600-query main-paper benchmark.

Statistic	Public (200)	Main paper (600)
count	200	600
mean	17.95	18.40
std	5.81	5.90
min	8	8
25%	13	14
50%	18	18
75%	22	23
90%	25	26
95%	27	28
max	35	37

Table 43: Query-length statistics (in whitespace-separated tokens) for the 200-query public release and the 600-query main-paper benchmark.

evaluator→optimizer loop; the final (head) plan is then reviewed and, if necessary, lightly edited by annotators.

All text fields in the release (queries, generated plans, reference plans, and lineages) are anonymized using a combination of rule-based and NER-based passes: client/account names, account IDs, policy names, agent and customer names, and PII (e.g., email addresses, phone numbers, SSN-like patterns, date-of-birth style dates) are replaced with abstract placeholders (e.g., `CLIENT_001`, `ACCOUNT_ID_007`, `PERSON_003`, `EMAIL_002`).

The release consists of two tables:

- **`queries.csv`** (200 rows): one row per query, with columns `query_id`, `Sample` (Train/Test), the natural-language `query`, binary flags `is_query_subjective` and `is_query_compound`, grouped best-plan length (`# steps in the BPP`, `# steps in the BPP - Grouped`), the human-edited reference plan (`best_plan`), and the plan lineage (`plan_lineage`). The latter two are stored as JSON strings that follow the plan schema in Sec. 2.

- **`plans.csv`** (5600 rows): one row per query–planner–prompt triple, with columns `query_id`, `llm`, `prompt_type`, and the generated plan (`plan`). We deliberately *omit* metric-wise and one-shot evaluator scores from this file: releasing scores computed by our proprietary judge stack would couple the public dataset too tightly to our internal evaluation implementation and could encourage overfitting to our specific scoring behavior. Instead, we intend this release to support independent assessment of the same planning task using alternative judges and metrics.

Although it is not the exact dataset used for the main quantitative results, this public set exposes concrete query–plan pairs, human-edited reference plans and lineages, and per-planner outputs for all 14 models and both prompt settings, enabling inspection of the task structure, planning behaviors, and potential biases in a way that closely mirrors the internal benchmark.

Representativeness of the 200-query release. Although the public release is built from a synthetic test account, we design it to mirror the main 600-query benchmark along the structural dimensions that drive planning difficulty. Table 42 shows that the marginal distributions over subjectivity (subjective vs. objective), compoundness (simple vs. compound), and grouped best-plan length ($[1, 2]$, $[3, 4]$, $[5, 15]$ steps) are very similar across the two datasets (e.g., subjective queries are 52.5% vs. 51.8%, simple queries 53.0% vs. 53.8%). Query-length statistics are also closely matched (mean 17.95 vs. 18.40 tokens; Table 43). As expected, the lexical distributions diverge somewhat (Tables 44–

Rank	Public (200)	Main paper (600)
1	calls (151)	calls (460)
2	customers (65)	orders (260)
3	last (61)	deliveries (210)
4	customer (50)	customer (180)
5	qa (49)	customers (185)
6	agents (48)	drivers (170)
7	call (45)	qa (150)
8	sentiment (43)	agents (145)
9	issues (37)	call (140)
10	during (36)	sentiment (130)

Table 44: Top unigrams (by frequency) in queries from the 200-query public release and the 600-query main-paper benchmark.

Rank	Public (200)	Main paper (600)
1	qa scores (25)	late deliveries (95)
2	customer sentiment (20)	missing items (88)
3	during calls (16)	customer complaints (82)
4	average qa (15)	qa scores (70)
5	last quarter (15)	customer sentiment (55)
6	many calls (15)	restaurant outages (52)
7	last month (14)	app crashes (48)
8	calls related (13)	during calls (45)
9	percentage calls (12)	last quarter (40)
10	calls tagged (12)	last month (38)

Table 45: Top bigrams (by frequency) in queries from the 200-query public release and the 600-query main-paper benchmark.

LLM	**With Lineage**			**Without Lineage**		
	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)	(*A+*) Extr. good, very good (%)	(*A*) Extr. good, very good, good (%)	(*B*) Extr. good, very good, good, acceptable (%)
o3-mini	44.44	54.9	74.77	**51.24**	67.97	82.61
gpt-4o	**46.83**	64	83.25	42.14	59.31	84.81
gpt-4o-mini	**32.46**	49.46	65.43	32.2	52.65	72.07
claude-3-5-haiku	24.67	45.32	61.94	**30.21**	47.33	72.01
claude-sonnet-4	30.88	53.53	74.63	**31.63**	52.53	78.26
llama4-maverick-17b-instruct	20.55	36.99	54.97	**21.06**	41.1	63.19
nova-pro	18.46	43.07	62.55	**19.99**	42.04	59.98
claude-3-7-sonnet	**30.85**	48.85	71.48	20.05	39.59	69.93
llama3-3-70b-instruct	**19.01**	44.69	60.62	17.47	35.96	55.99
gpt-4.1-nano	**17.41**	28.68	55.83	16.39	30.73	53.78
nova-micro	**20.89**	35.89	53.83	15.89	37.43	54.35
nova-lite	**14.02**	30.65	48.75	13.57	34.17	49.25
llama3-2-3b-instruct	5.09	10.69	17.31	**5.6**	11.71	26.98
llama3-2-1b-instruct	0	0	0	0	0	1.63
Grand Total (#)	**100**			**100**		

Table 46: Plan generation quality comparison using prompts **with and without lineage**, evaluated with the **one-shot evaluator** on the **test split of the public dataset**. The highest score in the **Extremely Good, Very Good** bucket is highlighted in green; blue indicates better performance with lineage, and magenta indicates better performance without lineage.

45): the public set focuses on a single synthetic account with generic contact-center terminology (e.g., *qa scores*, *customer sentiment*), whereas the internal benchmark covers multiple real accounts with account-specific drivers (e.g., *late deliveries*, *missing items*). This difference is desirable: it preserves

LLM	Overall [0-100]	Format [0-20]	Tool Prompt Align. [0-20]	Step Exec. [0-15]	Query Adhr. [0-15]	Depend. [0-10]	Redund. [0-10]	Tool Usage Compl. [0-10]
claude-3-7-sonnet	**86.27**	**18.77**	**15.57**	12.84	12.82	9.93	9.13	7.2
llama4-maverick-17b-instruct	83.25	17.34	14.16	13.26	12.85	9.45	9.52	6.67
gpt-4o	84.13	16.95	14.2	13.44	13.28	**9.95**	8.87	7.44
claude-sonnet-4	81.65	13.17	14.98	13.48	12.62	9.87	8.78	**8.73**
llama3-3-70b-instruct	80.55	17.2	14.66	13.68	12.22	9.64	9.46	3.7
nova-pro	79.67	15.59	14.65	**14.4**	12.38	9.86	9.42	3.37
nova-micro	78.54	18.54	13.63	13.14	12.28	9.72	9.6	1.61
gpt-4o-mini	78.52	14.28	14.77	13.81	12.55	9.13	9.27	4.71
gpt-4.1-nano	77.58	13.55	13.96	13.44	11.98	9.14	8.84	6.68
o3-mini	74.03	11.22	15.23	11.26	**14**	8.92	**9.62**	3.78
claude-3-5-haiku	70.11	12.42	12.57	11.4	11.18	8.3	8	6.23
nova-lite	68.38	13.43	12.54	12.18	11.56	8.14	9.12	1.42
llama3-2-3b-instruct	70.67	16.65	10.96	10.67	9.76	9.25	9.09	4.28
llama3-2-1b-instruct	21.6	0	0.07	0	11.2	4.48	2.39	3.45
Average (Normalized)	**73.93**	**71.11**	**64.98**	**79.53**	**81.28**	**90.00**	**86.50**	**49.49**

Table 47: Plan generation quality using prompts **with lineage**, evaluated with the **metric-wise evaluator** on the **test split of the public dataset**. The **Normalized Average** in the last row shows the average per metric normalized by that metric's maximum score. Highest scores per metric are highlighted in blue, and the three metrics with the lowest normalized scores are highlighted in red.

LLM	Overall [0-100]	Format [0-20]	Tool Prompt Align. [0-20]	Step Exec. [0-15]	Query Adhr. [0-15]	Depend. [0-10]	Redund. [0-10]	Tool Usage Compl. [0-10]
claude-3-7-sonnet	**85.08**	17.72	15.8	11.7	**13.65**	9.82	9.28	7.11
llama4-maverick-17b-instruct	83.81	17.62	13.8	12.87	12.55	9.91	8.99	8.06
gpt-4o	84.74	16.33	14.93	13.14	13.08	**9.96**	8.8	8.5
claude-sonnet-4	79.87	11.17	**17.06**	12.06	13.23	9.92	8.92	7.51
llama3-3-70b-instruct	81.97	15.4	14.74	13.46	12.47	9.69	9.18	7.03
nova-pro	82.96	15.97	14.31	14.15	12.73	9.81	8.89	7.1
nova-micro	83.11	**18.12**	13.41	13.61	12.29	9.61	8.77	7.29
gpt-4o-mini	84	17.03	13.92	**14.17**	12.45	9.75	8.67	8
gpt-4.1-nano	76.29	12.96	13.58	12.8	12.02	8.95	8.16	7.83
o3-mini	76.32	12.65	15.34	10.89	13.49	9.32	**9.72**	4.91
claude-3-5-haiku	80.93	14.47	15.71	13.82	11.94	9.66	9	6.33
nova-lite	72.85	14.81	12.7	12.65	11.53	8.62	9.01	3.54
llama3-2-3b-instruct	75.92	16.35	10.81	11.99	11.65	8.82	7.75	8.55
llama3-2-1b-instruct	60.7	9.64	9.67	9.47	8.08	7.55	6.8	**9.49**
Average (Normalized)	**79.18**	**75.09**	**69.92**	**84.18**	**81.50**	**94.00**	**87.08**	**72.31**

Table 48: Plan generation quality using prompts **without lineage**, evaluated with the **metric-wise evaluator** on the **test split of the public dataset**. The **Normalized Average** in the last row shows the average per metric normalized by that metric's maximum score. Highest scores per metric are highlighted in blue, and the three metrics with the lowest normalized scores are highlighted in red.

realistic contact-center vocabulary while avoiding leakage of proprietary account details, and our planning metrics depend primarily on plan structure rather than surface word frequencies.

Planner performance on the public subset. For completeness, we also re-run our one-shot and metric-wise evaluations on the test split (100 queries) of the 200-query public release. The resulting planner-level trends closely mirror those reported for the 600-query benchmark: the same models occupy the top tiers under both the one-shot evaluator and the metric-wise aggregate, and relative gaps across planners are very similar (cf. Tables 46, 47, and 48). This supports using the public subset as a small-scale proxy for the full benchmark.

E.3 Human Annotation Protocol and Agreement

Annotation targets. Human annotators (in-house team) produced: (i) gold (best-possible)

plans, (ii) gold labels for the *Step-wise Evaluator* (diagnostic tags per step) and *Plan Optimizer* (fully revised plan) used to supervise and tune the feedback loop; (iii) per-metric rankings for high-quality lineage plans (seven metrics), and (iv) 7-point overall quality labels for the last but one plan vs. last plan (gold plan) comparison in the lineage (*Extremely Bad* $\rightarrow$ *Extremely Good*).

Annotator configuration. Each item was independently labeled by two annotators. Disagreements were adjudicated by a third senior annotator across tasks. For metric-wise ranks and 7-point labels, we report raw agreement and inter-annotator agreement.

Annotation Guidelines.
General: Read the query and constraints carefully. Plans must be executable in principle (no tool execution required), use the correct tool for each datum, maintain a valid DAG of dependencies, and avoid redundancy. Prefer "interaction" over "call" unless the query explicitly says "call". Use placeholders consistently. Make sure that the standard placeholders (Refer D.1) are used correctly.

Gold Plan: Produce the best-possible, minimal plan that answers the query under stated constraints. Ensure each step is atomic, the tool choice is justified, and dependencies are complete and acyclic. Merge only when it reduces redundancy without harming executability.

Step-wise Evaluator (tags): For each step, assign applicable diagnostic tags from the closed set {INCORRECTTOOL, INCORRECTPROMPT, COMPLEXPROMPT, REPEATEDDETAIL, MULTITOOLPROMPT, NOCHANGE}. Tag only what is *clearly* violated; do not over-tag.

Plan Optimizer (revised plan): Apply local repairs first (Change 0), then global coherence fixes (Change 1) to preserve a valid DAG. Splits/merges are allowed if they improve clarity or executability. Do not introduce tool execution; modify *text* only.

Metric-wise Ranking (7 metrics): For the candidate plans in a lineage, rank or score each metric independently using the rubric. Do not force consistency across metrics; evaluate each in isolation.

7-point Overall Similarity (last but one vs. last (gold)): You are given a *gold* plan and your task is to compare the degree of closeness between the plan to be scored (*last but one plan* in the lineage) and the *gold* plan based on (i) Precision: What % of steps in the plan to be scored are present in the gold plan; (ii) Recall: What % of steps in the gold plan are present in the plan to be scored; (iii) F1-score; (iv) Format errors: Independently assess the plan to be scored for the presence of format-related errors. Use the 7-level rubric thresholds consistently to make a final decision based on the aforementioned dimensions.

Agreement metrics. We report:

- **Cohen's** κ (Cohen, 1960) for two-rater nominal decisions (per-metric rank selections treated as nominal within each query).
- **Quadratic-Weighted Kappa (QWK)** (Cohen, 1968) for the 7-point ordinal overall rating.
- **Fleiss'** κ (Fleiss, 1971) only if more than two raters are used on a subset (not the case here; placeholder shown for completeness).

Formulas. For two raters with observed agreement p_o and chance agreement p_e:

$$\kappa = \frac{p_o - p_e}{1 - p_e}.$$

For ordinal categories $c \in \{1, \ldots, K\}$ with weight matrix $W_{ij} = 1 - \left(\frac{i-j}{K-1}\right)^2$ (quadratic weights), and empirical rating matrix $\mathbf{O}$ and expected $\mathbf{E}$:

$$\kappa_w = 1 - \frac{\sum_{i,j} W_{ij}\, O_{ij}}{\sum_{i,j} W_{ij}\, E_{ij}}.$$

For $m > 2$ raters, Fleiss' κ is computed over category proportions per item (Fleiss, 1971).

Confidence intervals. We report 95% CIs via nonparametric bootstrap (1,000 resamples over items). We also report macro-averaged κ across metrics (for the seven per-metric ranks) and micro-averaged (pooled items).

E.3.1 Results

We employed two independent annotators for all reliability measurements. For nominal labels we report Cohen's κ; for the 7-point ordinal similarity rating returned by the one-shot Judge we report linearly *weighted* κ. We also include raw percent agreement for context.

Methodological details. For *Gold Plan* and *Plan Optimizer (7-pt sim)*, each item yields two independent plans A and B from two independent annotators. We compute a *symmetric* similarity as follows: (i) run the one-shot evaluator twice, $A \rightarrow B$

Task	Unit	#Items	Agreement (%)	κ	95% CI
Metric-wise Evaluator	plan×metric	1680	90.9	0.82	[0.79, 0.85]
One-shot Evaluator (7-pt sim)	query (ordinal)	80	86.4	0.74_{w}	[0.68, 0.79]
Gold Plan (7-pt sim)	query (ordinal)	80	83.1	0.70_{w}	[0.64, 0.76]
Step-wise Evaluator (tags)	step (multi-label)†	400	84.6	$0.68^{‡}$	[0.64, 0.72]
Plan Optimizer (7-pt sim)	revision (ordinal)	160	80.2	0.66_{w}	[0.60, 0.72]

Table 49: Two-annotator reliability on the validation set (80 queries). For ordinal tasks we use linearly weighted Cohen's κ.

Notes. κ is Cohen's κ (linearly weighted for the ordinal 7-point similarity tasks). Similarity-based rows use the tuned one-shot evaluator to score closeness between alternative plans.
† **Step-wise Evaluator (multi-label):** Per-tag κ computed and macro-averaged across *complex prompt, incorrect prompt, incorrect tool, repeated detail, multi-tool prompt, no change*; Computed over $80 \times \bar{s}$ steps with $\bar{s}$=5.
‡ Macro-averaged across tags.

and $B \to A$; (ii) average the directional F1 scores to obtain $s_{\text{F1}} \in [0, 1]$;[7] (iii) set a binary format flag to 1 only if *both* plans pass format checks (`format_ok`(A)∧`format_ok`(B)); and (iv) define the scalar similarity s by combining the format flag with s_{F1} (e.g., clamping to the lowest bin if format fails, else using s_{F1}). We discretize s into the 7-point ordinal label via the same fixed thresholds used in our one-shot evaluator prompts (Refer Table 33). We then compute linearly weighted Cohen's κ on these ordinal labels. When similarity $<$ *Very Good*, a senior annotator adjudicates and their decision defines the gold used in subsequent experiments.

Interpretation. Agreement is highest for rubric-driven metric-wise labels, moderate for holistic ordinal judgments (one-shot, gold, step-wise evaluator), and lowest for plan optimizer similarity, reflecting the ambiguity of the different tasks. Overall, all scores fall in the *substantial* range or higher across tasks as per Landis and Koch, 1977.

E.4 Judge Selection and Cross-Judge Robustness

Validation-time human alignment (metric-wise evaluator). In the main paper we validated the metric-wise evaluator using Claude-Sonnet-4 as judge, comparing relaxed triplet rankings against human-annotated inequalities across seven metrics on the 80-query validation set (Tab. 21). To test robustness to the choice of judge, we repeated this analysis with GPT-5 in the same *reference-based, deconstructed* configuration. Table 21 summarizes percent-correct inequalities for both judges. For Sonnet-4, the reference-based, deconstructed setting achieves $> 80\%$ agreement on all metrics and $> 90\%$ on DEPENDENCY, FORMAT, REDUNDANCY, and TOOL USAGE COMPLETENESS. GPT-5 in the same configuration is comparably strong, often slightly better on FORMAT (99.23%), REDUNDANCY (90.82%), and QUERY ADHERENCE (87.38%), and slightly weaker on STEP EXECUTABILITY (65.70% vs. 79.43%). Importantly, both judges achieve 94.12% agreement on TOOL USAGE COMPLETENESS. These results indicate that our metric-wise rubric is learnable by multiple judge models from different families.

Validation-time human alignment (one-shot evaluator). We also compared Sonnet-4 and GPT-5 as one-shot overall judges on the same validation set. Tables 22 and 23 report per-label and macro Precision/Recall/F1 across the seven quality tags (Extremely Bad → Extremely Good). Both judges show strong agreement with annotators, but Sonnet-4 achieves higher macro F1 (0.921) than GPT-5 (0.882), motivating our choice of Sonnet-4 as the primary judge in the main experiments.

Cross-judge consistency on the test subset. To quantify how much planner rankings depend on the choice of judge, we re-scored a 50-query subset of the test split per planner (14 planner LLMs, prompt type = *Without Lineage*) with GPT-5, in addition to the existing Sonnet-4 scores. For each judge we computed per-planner mean metric-wise totals and per-metric means, as well as mean one-shot quality scores, then ranked planners in descending order of mean score. We see that the same set of strong planners occupy the top tier under both judges (e.g., GPT-4o, GPT-4o-mini, Claude-3-7-Sonnet, Llama3-3-70B, Nova-Pro/Micro), while Claude-Sonnet-4 itself is mid-pack as a planner under both

[7] Under symmetric matching, this equals the undirected set-F1 $2|A \cap B|/(|A| + |B|)$; the bi-directional averaging adds robustness to minor directional heuristics.

judges. Table 24 reports Spearman rank correlations between Sonnet-4 and GPT-5 across metrics: we observe ρ=0.60 (p=0.023) for the overall metric-wise score, ρ=0.52 for the one-shot quality score, and high correlations for key structural metrics (e.g., DEPENDENCY ρ=0.84, QUERY ADHERENCE ρ=0.79, TOOL–PROMPT ALIGNMENT ρ=0.72). TOOL USAGE COMPLETENESS exhibits a lower cross-judge correlation (ρ=0.35). This metric is only defined for sub-queries that *should* use both T2S and RAG; for queries where no sub-step requires both tools, the score is NA. As a result, it is based on comparatively few (edge-case) instances, even though each judge individually attains 94.12% agreement with humans on this metric.

Takeaway. Taken together, these results show that (i) multiple judge models from different families can be calibrated to our rubric and agree well with humans, and (ii) planner rankings and per-metric assessments are broadly consistent across Sonnet-4 and GPT-5. This directly mitigates concerns about strong model-family bias or prompt-specific artifacts: Sonnet-4 is a slightly stronger one-shot judge, but our main comparative conclusions about planner quality do not hinge on this particular choice.

E.5 Models and Prompts

Plan generation (14 LLMs). We evaluate the following models: *Claude-3-7-Sonnet, Claude-Sonnet-4, Claude-3-5-Haiku, Nova-Pro, Nova-Lite, Nova-Micro, Llama3-2-1B-Instruct, Llama3-2-3B-Instruct, Llama3-70B-Instruct, Llama4-Maverick-17B-Instruct, GPT-4o, GPT-4o-Mini, GPT-4.1-Nano, o3-Mini (medium reasoning)*. Two prompts are used per query: (*i*) **without lineage** (task + tool specs + few-shot exemplars) and (*ii*) **with lineage** (adds per-query lineage exemplars in the reference section). Full prompt templates are in Table 27.

Evaluation LLMs. We use *Claude-Sonnet-4* as the Judge LLM for both evaluation modes:

1. **Metric-wise evaluators** (seven specialist rubrics; Sec. 4, Appx. D.1).
2. **One-shot overall evaluator** (Precision/Recall/F_1, Format, Dependencies, Placeholders, 7-point rating; Appx. D.2).

Feedback-loop modules. The Step-wise Evaluator and Plan Optimizer (Appx. C.4) are both instantiated with *GPT-4o*. We set `max_passes`=**4** by default; this provided a favorable accuracy/latency trade-off in pilot runs. We cap per-pass budget (LLM calls and wall time) to avoid degenerate loops.

E.6 Decoding, Seeds, and Hyperparameters

We apply deterministic decoding for evaluation LLMs (temperature $\in \{0\}$; top-p = 1.0) and low-to-moderate temperature for generators (per-model defaults; tabulated below). All runs fix random seeds. We enable response validation (JSON parsing) with retry-on-format for evaluators.

E.7 Infrastructure and Reproducibility

We implement all components in Python. Calls are routed via Bedrock and LiteLLM with exponential backoff and per-provider rate limits. We cache intermediate LLM outputs keyed by (model, prompt, seed) and persist full artifacts (plans, lineages, evaluator JSON) for auditability. The model configurations used across modules are provided in Table 50.

E.8 Additional Controls

We enforce (i) JSON validation with structured error messages to prompt repair; (ii) strict placeholder checks; (iii) timeouts and retries; and (iv) lineage provenance (every edited plan version is stored with a diff and reason tag from the Evaluator).

F Grouped Analyses for Plan Generation Quality

F.1 Objective vs. Subjective

One-shot (Tables 5–6) With lineage: 7/14 models have higher *A*+ proportions on Objective than Subjective; 6/14 are lower; 1 unchanged. Without lineage: 9/14 higher on Objective, 4/14 lower, 1 unchanged. *Interpretation:* modest, model-dependent edge for Objective in the no-lineage setting; not conclusive overall.

Metric-wise (Table 13) With lineage: 11/14 models score higher on Subjective than Objective; without lineage: 9/14 higher on Subjective. *Interpretation:* metric decomposition favors Subjective for many models, despite one-shot results showing only a weak/no advantage for Objective.

F.2 Simple vs. Compound

One-shot (Tables 7–8) With lineage: 12/14 higher *A*+ for Simple; without lineage: 13/14

Module	Model	Source	Temperature	Top-p	Max Tokens
Query Gen	GPT-4o	Azure	0.2	1.0	4096
Plan Gen (all 14)	(varies)	Azure, Bedrock	0.2	1.0	4096
Metric-wise Eval	Claude-Sonnet-4	Bedrock	0.0	1.0	4096
One-shot Judge	Claude-Sonnet-4	Bedrock	0.0	1.0	4096
Step-wise Eval	GPT-4o	Bedrock	0.0	1.0	4096
Plan Optimizer	GPT-4o	Bedrock	0.0	1.0	4096

Table 50: LLM configurations used across modules

Metric	Value
Average # distinct plans / lineage	5.5
Average # passes / query	3.0
Average # revisions / pass (accepted)	1.5
Average # steps / initial plan	5.4
Average # steps / best plan	4.8
Average # distinct tool types / initial plan	2.4
Average # distinct tool types / best plan	2.2

Table 51: Summary statistics of the iterative evaluator→optimizer loop (validation split). "Revisions/pass" counts *accepted* edits.

higher for Simple. *Interpretation:* strong and consistent advantage for Simple queries.

Metric-wise (Table 14) With lineage: 11/14 higher overall for Simple; without lineage: 10/14 higher. *Interpretation:* mirrors one-shot; complexity hurts.

F.3 Plan Length: $[1, 4]$ vs. $[5, 15]$ Steps in the Best Possible Plans

One-shot (Tables 9–10) With lineage: 13/14 higher $A+$ for $[1, 4]$; without lineage: 13/14 higher for $[1, 4]$. *Interpretation:* shorter gold plans are markedly easier to match.

Metric-wise (Table 16) With lineage: 11/14 higher overall for $[1, 4]$; without lineage: 12/14 higher. *Interpretation:* consistent with one-shot; longer gold plans expose weaknesses in tool assignment and dependency wiring.

F.4 Number of Hops: 0/1/2/3+

One-shot (Tables 11–12) With lineage: 10/14 show higher $A+$ for One-Hop (vs. Two-Hop), while 3/14 favor Two-Hop. Without lineage: 6/14 favor One-Hop, 7/14 favor Two-Hop. *Interpretation:* no stable, cross-model pattern.

Metric-wise (Table 15) With lineage: 6/14 higher overall for One-Hop; 4/14 higher for Two-Hop. Without lineage: 9/14 higher for One-Hop; 3/14 higher for Two-Hop. *Interpretation:* small, inconsistent edges - hops alone are not a robust predictor.

Summary. Across grouped factors, **query simplicity** and **shorter best-plan length** correlate most strongly with better quality. The effects of **lineage prompting** and **hops** are mixed and model dependent.

G Grouped Analyses: Effectiveness of the Evaluator→Optimizer Loop

G.1 Objective vs. Subjective

Objective (Table 18) *Extremely Good* increases from **4.15%** (pre) to **9.54%** (post); *Very Good* from **2.07%** to **10.37%**. *Observation:* substantial uplift on top brackets; loop strongly benefits precision-oriented, verifiable tasks.

Subjective (Table 18) *Extremely Good* moves from **5.02%** (pre) to **6.95%** (post); *Very Good* from **15.83%** to **18.53%**. *Observation:* consistent, moderate improvement; gains smaller than Objective but still meaningful.

G.2 Simple vs. Compound

Simple (Table 17) *Extremely Good* improves from **5.58%** (pre) to **11.15%** (post); *Very Good* from **11.15%** to **17.84%**. *Observation:* total uplift of ∼**12.26pp** across top two buckets; loop is highly effective when plans are short and atomic.

Compound (Table 17) *Extremely Good* rises from **3.46%** (pre) to **4.76%** (post); *Very Good* from **6.49%** to **10.82%**. *Observation:* clear improvement (∼**5.63pp**); smaller than Simple, but still positive given higher compositional load.

G.3 Hop Count (0/1/2/3+)

Zero and One hop (Table 19) **0-hop**: *Extremely Good* stays at **7.95%**, while *Very Good* increases from **0.0%** to **9.09%**. **1-hop**: *Extremely Good* from **4.93%** (pre) to **9.85%** (post); *Very Good* from **13.79%** to **18.72%**. *Observation:* noticeable gains even for short-hop plans, particularly in the *Very Good* bracket.

Two and Three-plus hops (Table 20) **2-hop**: *Extremely Good* from **3.57%** to **7.86%**; *Very Good* from **9.29%** to **12.86%**. **3+ hop**: *Extremely Good* from **0.00%** to **4.35%**; *Very Good* from **7.25%** to **14.49%**. *Observation:* largest *relative* uplifts are in **3+ hop** and **2-hop** settings, indicating the loop is most beneficial when plans require longer, more error-prone compositions.

Summary. Across all strata, the *Evaluator→Optimizer* loop consistently shifts mass from mid/low buckets into *Very Good*/*Extremely Good*, with the strongest relative effects for *Objective*, *Simple*, and *3+ hop* cases.

H Module Validation on Validation Set (N=80): Full Results

H.1 Metric-wise Evaluator Validation

Setup recap. For each validation query, we take the final three plans in its lineage (highest quality, with the last being the human-verified best). Humans annotate *per-metric* and *overall* orderings among these three plans (relaxed inequality). The LLM evaluators score the same triple; we convert scores to an ordering and compute *relaxed triplet ranking agreement* (match if human $a \leq b \leq c$ is predicted as $a \leq b \leq c$, allowing ties).

Method. We evaluate two designs: (1) *Single*: a unified prompt defines all seven metrics and requests unweighted metric scores + rationale. (2) *Deconstructed*: seven specialized prompts, one per metric. Each design is run in *reference-free* (query + plan) and *reference-based* (query + plan + gold plan) modes. For each metric, we compute the percent of correct triplet inequalities across the 80 queries.

Results. Table 21 reports agreement by metric and setting. The *deconstructed, reference-based* variant dominates, exceeding 90% agreement for DEPENDENCY, FORMAT, TOOL USAGE COMPLETENESS and surpassing 80% for QUERY ADHERENCE, REDUNDANCY, TOOL-PROMPT ALIGNMENT. STEP EXECUTABILITY achieves 79.43%. Reference-free settings trail as expected; *Single* underperforms *Deconstructed* across the board, indicating reduced interference when judging metrics in isolation.

H.2 One-Shot Overall Evaluator Validation

Method. The judge LLM (Appendix E.5) labels each plan among seven categories (*Extremely Bad* to *Extremely Good*). We compute per-class Precision/Recall/F1 against human labels and macro averages.

Results. Table 22 shows macro Precision/Recall/F1 of $0.92/0.93/0.92$. All classes have F1 ≥ 0.85, supporting reliable downstream use of the judge for plan quality decisions.

H.3 Step-Wise Evaluator Validation

Method. On N=400 step instances (80 queries $\times$ ≈5 steps/plan), we compute multi-label Precision/Recall/F1 across tags: NO CHANGE, INCORRECT TOOL, INCORRECT PROMPT, COMPLEX PROMPT, REPEATED DETAIL, MULTI-TOOL PROMPT by comparing Judge LLM labels against human labels.

Results. Table 25 reports macro F1 = 0.84. INCORRECT PROMPT and REPEATED DETAIL reach 0.91 F1, indicating the evaluator reliably flags prompt-level errors and SQL redundancy. NO CHANGE is most challenging (F1 = 0.75), reflecting conservative behavior when judging step sufficiency.

H.4 Plan Optimizer Validation

Method. For 160 plan pairs (80 queries $\times$ 2 revisions per query), we compare the optimizer's revision against the human gold using the tuned one-shot judge (Table 33). We report the distribution across seven quality tags.

Results. As shown in Table 26, 74.5% of optimizer outputs fall into *Good* or better (*Extremely Good* 28.13%, *Very Good* 24.38%, *Good* 21.88%), and 10.63% are *Acceptable*; the remainder capture difficult edits where structural rewrites are needed. This supports using the loop to automatically lift initial plans to near-usable quality.

Interpretation. (i) Per-metric judgments are most stable when isolated and provided the gold plan (deconstructed, reference-based). (ii) The one-shot judge is precise enough for automatic triage. (iii) Step-wise tags are discriminative for prompt/tool issues and redundancy. (iv) The optimizer

substantially improves plan quality relative to initial drafts, aligning with Sec. 6.2.